Karunya INSTITUTE OF TECHNOLOGY AND SCIENCES
(Declared as Deemed to be University under Sec.3 of the UGC Act, 1956)
MoE, UGC & AICTE Approved
NAAC A++ Accredited

DIVISION OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTER SCIENCE AND TECHNOLOGY

LABORATORY RECORD

ODD SEMESTER 2023-2024

Name : **Bewin Felix R A**

Reg.No : **URK21CS1128**

Course Code : **20CS2031L**

Course Name : **Introduction to Data Science**

NOVEMBER 2023

Karunya INSTITUTE OF TECHNOLOGY AND SCIENCES

(Declared as Deemed to be University under Sec.3 of the UGC Act, 1956)

MoE, UGC & AICTE Approved

NAAC A++ Accredited

DIVISION OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTER SCIENCE AND TECHNOLOGY

LABORATORY RECORD

Academic Year 2023-2024

Course Code

20CS2031L

Course Name

Introduction to Data Science

Register No. URK21CS1128

It is hereby certified that this is the bonafide record of work done by Mr./Ms. Bewin Felix R A during the odd semester of the academic year 2023-2024 and submitted for the University Practical Examination held on 14.11.2023.

Faculty-in-charge **Program Coordinator**

Name:

Examiner

TABLE OF CONTENTS

Ex.1 Working with Python Data Structures URK21CS1128

September 4, 2023

AIM: This course aims to teach learners how to work with Python data structures in data science, covering concepts, manipulation, and visualization.

DESCRIPTION: The course covers Python data structures like lists, tuples, dictionaries, sets, and arrays, along with NumPy and Pandas for data analysis. Participants will learn to visualize data and apply structures to real-world data science problems.

1. Create an empty dictionary. Fill the dictionary with prod_code and prod_name as pair by user input (Use prod_code as a key). Take one prod_code as input from the user and traverse through dictionary to find the corresponding prod_name and display the same.

[1]:
```
#1128
empty_dict = {}

n = int(input("Enter the number of product pairs you want to add: "))
for _ in range(n):
    prod_code = input("Enter the product code: ")
    prod_name = input("Enter the product name: ")
    empty_dict[prod_code] = prod_name

user_input_code = input("Enter the product code to find the corresponding␣
↪product name: ")
if user_input_code in empty_dict:
    print("Corresponding product name:", empty_dict[user_input_code])
else:
    print("Product code not found in the dictionary.")
```

```
Enter the number of product pairs you want to add:  2
Enter the product code:  project
Enter the product name:  book
Enter the product code:  session
Enter the product name:  guide
Enter the product code to find the corresponding product name:  session

Corresponding product name: guide
```

2. Create an empty list. Fill the list with strings by getting user input. Find the list of words that are longer than n from a given list of words. Sample List : [‘the’,’quick’,’brown’,’fox’] n : 3

[2]:
```python
#1128
empty_list = []

n = int(input("Enter the number of words you want to add to the list: "))
for _ in range(n):
    word = input("Enter a word: ")
    empty_list.append(word)

min_length = int(input("Enter the minimum word length to filter the words: "))
result_list = [word for word in empty_list if len(word) > min_length]
print("Words longer than", min_length, ":", result_list)
```

```
Enter the number of words you want to add to the list:  3
Enter a word:  is
Enter a word:  was
Enter a word:  the
Enter the minimum word length to filter the words:  2

Words longer than 2 : ['was', 'the']
```

3. Create an empty set. Fill the set with values by getting user input. Check if a given value is present in a set or not.

[3]:
```python
#1128
empty_set = set()

n = int(input("Enter the number of values you want to add to the set: "))
for _ in range(n):
    value = input("Enter a value: ")
    empty_set.add(value)

search_value = input("Enter the value to check if it's present in the set: ")
if search_value in empty_set:
    print(search_value, "is present in the set.")
else:
    print(search_value, "is not present in the set.")
```

```
Enter the number of values you want to add to the set:  3
Enter a value:  2
Enter a value:  3
Enter a value:  1
Enter the value to check if it's present in the set:  2

2 is present in the set.
```

4. Create an empty tuple. Populate the tuple with values by getting user input. Count the occurrence of a given input number in the tuple. Sample : (50, 10, 60, 70, 50) n : 50

[4]:
```python
#1128
empty_tuple = ()
```

```
n = int(input("Enter the number of elements you want to add to the tuple: "))
for _ in range(n):
    value = int(input("Enter a number: "))
    empty_tuple += (value,)

search_number = int(input("Enter the number to count its occurrences in the␣
 ↪tuple: "))
count = empty_tuple.count(search_number)
print("Occurrences of", search_number, "in the tuple:", count)
```

```
Enter the number of elements you want to add to the tuple:  3
Enter a number:  1
Enter a number:  2
Enter a number:  3
Enter the number to count its occurrences in the tuple:  1

Occurrences of 1 in the tuple: 1
```

5. Create a 2D array and perform matrix subtraction using numpy.

[8]:
```
#1128
import numpy as np
array1 = np.array([[1, 2], [3, 4]])
array2 = np.array([[2, 1], [4, 3]])

result_array = array1 - array2
print("Result of matrix subtraction:")
print(result_array)
```

```
Result of matrix subtraction:
[[-1  1]
 [-1  1]]
```

6. Create a 2D array using numpy and find the maximum element in the matrix.

[9]:
```
#1128
import numpy as np
matrix = np.array([[5, 10, 15], [20, 25, 30], [35, 40, 45]])

max_element = np.max(matrix)
print("Maximum element in the matrix:", max_element)
```

```
Maximum element in the matrix: 45
```

7. Download the dataset from https://www.kaggle.com/datasets/varshamannem/toyato. Read the Toyota.csv file and display the basic details.

a. Display the top 10 rows
b. Display the last 5 rows
c. Display row and column details

d. Display size, shape, dimension and information summary

[6]:
```python
#1128
import pandas as pd

# Assuming you have already downloaded and placed the Toyota.csv file in the
↪current directory
file_path = "Toyota.csv"

# Read the CSV file into a DataFrame
df = pd.read_csv(file_path)

# a. Display the top 10 rows
print("Top 10 rows:")
print(df.head(10))

# b. Display the last 5 rows
print("\nLast 5 rows:")
print(df.tail())

# c. Display row and column details
print("\nRow and Column details:")
print("Number of rows:", df.index)
print("Number of columns:", df.columns)

# d. Display size, shape, dimension, and information summary
print("\nSize of the DataFrame:", df.size)
print("Shape of the DataFrame:", df.shape)
print("Number of dimensions:", df.ndim)

# Information summary
print("\nInformation summary:")
print(df.info())
```

```
Top 10 rows:
   Unnamed: 0  Price   Age     KM FuelType    HP  MetColor  Automatic    CC  \
0           0  13500  23.0  46986   Diesel    90       1.0          0  2000
1           1  13750  23.0  72937   Diesel    90       1.0          0  2000
2           2  13950  24.0  41711   Diesel    90       NaN          0  2000
3           3  14950  26.0  48000   Diesel    90       0.0          0  2000
4           4  13750  30.0  38500   Diesel    90       0.0          0  2000
5           5  12950  32.0  61000   Diesel    90       0.0          0  2000
6           6  16900  27.0     ??   Diesel  ????       NaN          0  2000
7           7  18600  30.0  75889      NaN    90       1.0          0  2000
8           8  21500  27.0  19700   Petrol   192       0.0          0  1800
9           9  12950  23.0  71138   Diesel  ????       NaN          0  1900

   Doors  Weight
```

```
0  three    1165
1      3    1165
2      3    1165
3      3    1165
4      3    1170
5      3    1170
6      3    1245
7      3    1245
8      3    1185
9      3    1105

Last 5 rows:
      Unnamed: 0  Price   Age     KM FuelType   HP  MetColor  Automatic    CC  \
1431        1431   7500   NaN  20544   Petrol   86       1.0          0  1300
1432        1432  10845  72.0     ??   Petrol   86       0.0          0  1300
1433        1433   8500   NaN  17016   Petrol   86       0.0          0  1300
1434        1434   7250  70.0     ??      NaN   86       1.0          0  1300
1435        1435   6950  76.0      1   Petrol  110       0.0          0  1600

     Doors  Weight
1431     3    1025
1432     3    1015
1433     3    1015
1434     3    1015
1435     5    1114

Row and Column details:
Number of rows: RangeIndex(start=0, stop=1436, step=1)
Number of columns: Index(['Unnamed: 0', 'Price', 'Age', 'KM', 'FuelType', 'HP',
'MetColor',
       'Automatic', 'CC', 'Doors', 'Weight'],
      dtype='object')

Size of the DataFrame: 15796
Shape of the DataFrame: (1436, 11)
Number of dimensions: 2

Information summary:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1436 entries, 0 to 1435
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  1436 non-null   int64
 1   Price       1436 non-null   int64
 2   Age         1336 non-null   float64
 3   KM          1436 non-null   object
 4   FuelType    1336 non-null   object
```

```
 5   HP          1436 non-null   object
 6   MetColor    1286 non-null   float64
 7   Automatic   1436 non-null   int64
 8   CC          1436 non-null   int64
 9   Doors       1436 non-null   object
 10  Weight      1436 non-null   int64
dtypes: float64(2), int64(5), object(4)
memory usage: 123.5+ KB
None
```

Result: The basic functionalities of data visualization using python were executed successfully.

Ex.2 Working with Data using Pandas

September 4, 2023

Expt: No 2

URK21CS1128

Bewin Felix R A

[ ]:
```
Aim: To execute the basic functionalities using pandas with data.
```

Description: Python Pandas is defined as an open-source library that provides high-performance data manipulation in Python. Started by Wes McKinney in 2008 out of a need for a powerful and flexible quantitative analysis tool, panda has grown into one of the most popular Python libraries. Pandas is built on top of the Numpy package, means Numpy is required for operating the Pandas.Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns. Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data.

Program:

[1]:
```
import pandas as pd
df = pd.read_csv("Titanic.csv")
df
```

[1]:
```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           …         …     …
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                  Name     Sex   Age  SibSp  \
0                              Braund, Mr. Owen Harris    male  22.0      1
1    Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
```

```
2                              Heikkinen, Miss. Laina  female  26.0      0
3        Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                            Allen, Mr. William Henry    male  35.0      0
..                                                ...     ...   ...    ...
886                             Montvila, Rev. Juozas    male  27.0      0
887                      Graham, Miss. Margaret Edith  female  19.0      0
888          Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1
889                             Behr, Mr. Karl Howell    male  26.0      0
890                               Dooley, Mr. Patrick    male  32.0      0

     Parch            Ticket     Fare Cabin Embarked
0        0         A/5 21171   7.2500   NaN        S
1        0          PC 17599  71.2833   C85        C
2        0  STON/O2. 3101282   7.9250   NaN        S
3        0            113803  53.1000  C123        S
4        0            373450   8.0500   NaN        S
..     ...               ...      ...   ...      ...
886      0            211536  13.0000   NaN        S
887      0            112053  30.0000   B42        S
888      2        W./C. 6607  23.4500   NaN        S
889      0            111369  30.0000  C148        C
890      0            370376   7.7500   NaN        Q

[891 rows x 12 columns]
```

Q1: Display the columns that have null and its count

[5]:
```
print("URK21CS1128")
df0=df.isnull().sum().sum()
print("Count:",df0)
```

```
URK21CS1128
Count: 866
```

[6]:
```
df
```

[6]:
```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3
```

```
                                                  Name     Sex   Age  SibSp  \
0                              Braund, Mr. Owen Harris    male  22.0      1
1    Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                               Heikkinen, Miss. Laina  female  26.0      0
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                             Allen, Mr. William Henry    male  35.0      0
..                                                 …     …     …    …
886                              Montvila, Rev. Juozas    male  27.0      0
887                       Graham, Miss. Margaret Edith  female  19.0      0
888           Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1
889                              Behr, Mr. Karl Howell    male  26.0      0
890                                Dooley, Mr. Patrick    male  32.0      0

     Parch            Ticket     Fare Cabin Embarked
0        0         A/5 21171   7.2500   NaN        S
1        0          PC 17599  71.2833   C85        C
2        0  STON/O2. 3101282   7.9250   NaN        S
3        0            113803  53.1000  C123        S
4        0            373450   8.0500   NaN        S
..     …               …      …   …      …
886      0            211536  13.0000   NaN        S
887      0            112053  30.0000   B42        S
888      2        W./C. 6607  23.4500   NaN        S
889      0            111369  30.0000  C148        C
890      0            370376   7.7500   NaN        Q

[891 rows x 12 columns]
```

Q2: Display the statistical description of the numerical and non-numerical columns

[7]:
```python
print("URK21CS1128")
df.describe()
```

```
URK21CS1128
```

[7]:
```
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
```

```
mean      0.381594   32.204208
std       0.806057   49.693429
min       0.000000    0.000000
25%       0.000000    7.910400
50%       0.000000   14.454200
75%       0.000000   31.000000
max       6.000000  512.329200
```

[8]:
```python
print("URK21CS1128")
df.describe(include='object')
```

```
URK21CS1128
```

[8]:
```
                           Name   Sex  Ticket    Cabin Embarked
count                       891   891     891      204      889
unique                      891     2     681      147        3
top     Braund, Mr. Owen Harris  male  347082  B96 B98        S
freq                          1   577       7        4      644
```

Q3: Display the rows that 'Sex' column value is male and observe the count

[9]:
```python
print("URK21CS1128")
df2 = df[df['Sex']=='male']
df2
```

```
URK21CS1128
```

[9]:
```
     PassengerId  Survived  Pclass                            Name   Sex  \
0              1         0       3         Braund, Mr. Owen Harris  male
4              5         0       3        Allen, Mr. William Henry  male
5              6         0       3                Moran, Mr. James  male
6              7         0       1         McCarthy, Mr. Timothy J  male
7              8         0       3  Palsson, Master. Gosta Leonard  male
..           …         …       …                               …     …
883          884         0       2    Banfield, Mr. Frederick James  male
884          885         0       3           Sutehall, Mr. Henry Jr  male
886          887         0       2            Montvila, Rev. Juozas  male
889          890         1       1             Behr, Mr. Karl Howell  male
890          891         0       3              Dooley, Mr. Patrick  male

      Age  SibSp  Parch            Ticket     Fare Cabin Embarked
0    22.0      1      0         A/5 21171   7.2500   NaN        S
4    35.0      0      0            373450   8.0500   NaN        S
5     NaN      0      0            330877   8.4583   NaN        Q
6    54.0      0      0             17463  51.8625   E46        S
7     2.0      3      1            349909  21.0750   NaN        S
..    …     …      …                 …       …     …        …
883  28.0      0      0  C.A./SOTON 34068  10.5000   NaN        S
884  25.0      0      0    SOTON/OQ 392076   7.0500   NaN        S
```

```
886  27.0      0      0          211536  13.0000   NaN        S
889  26.0      0      0          111369  30.0000  C148        C
890  32.0      0      0          370376   7.7500   NaN        Q

[577 rows x 12 columns]
```

[10]:
```
#URK21CS1128
df2.shape[0]
```

[10]:
```
577
```

Q4: Display the Name and Age of first 25 rows with 'Embarked' column is C (Cherbourg)

[11]:
```
print("URK21CS1128")
df4 = df[df['Embarked']=="C"]
df4 = df4.head(25)[['Name','Age']]
df4
```

```
URK21CS1128
```

[11]:

	Name	Age
1	Cumings, Mrs. John Bradley (Florence Briggs Th…	38.0
9	Nasser, Mrs. Nicholas (Adele Achem)	14.0
19	Masselmani, Mrs. Fatima	NaN
26	Emir, Mr. Farred Chehab	NaN
30	Uruchurtu, Don. Manuel E	40.0
31	Spencer, Mrs. William Augustus (Marie Eugenie)	NaN
34	Meyer, Mr. Edgar Joseph	28.0
36	Mamee, Mr. Hanna	NaN
39	Nicola-Yarred, Miss. Jamila	14.0
42	Kraeff, Mr. Theodor	NaN
43	Laroche, Miss. Simonne Marie Anne Andree	3.0
48	Samaan, Mr. Youssef	NaN
52	Harper, Mrs. Henry Sleeper (Myna Haxtun)	49.0
54	Ostby, Mr. Engelhart Cornelius	65.0
57	Novel, Mr. Mansouer	28.5
60	Sirayanian, Mr. Orsen	22.0
64	Stewart, Mr. Albert A	NaN
65	Moubarek, Master. Gerios	NaN
73	Chronopoulos, Mr. Apostolos	26.0
96	Goldschmidt, Mr. George B	71.0
97	Greenfield, Mr. William Bertram	23.0
111	Zabour, Miss. Hileni	14.5
114	Attalah, Miss. Malake	17.0
118	Baxter, Mr. Quigg Edmond	24.0
122	Nasser, Mr. Nicholas	32.5

Q5: Display the rows that Age>20 and Survived status is 0

[12]:
```
print("URK21CS1128")
df5=df[(df.Age>20)&(df.Survived==0)]
df5
```

```
URK21CS1128
```

[12]:
```
     PassengerId  Survived  Pclass  \
0              1         0       3
4              5         0       3
6              7         0       1
13            14         0       3
18            19         0       3
..           ...       ...     ...
883          884         0       2
884          885         0       3
885          886         0       3
886          887         0       2
890          891         0       3

                                                  Name     Sex   Age  SibSp  \
0                              Braund, Mr. Owen Harris    male  22.0      1
4                             Allen, Mr. William Henry    male  35.0      0
6                              McCarthy, Mr. Timothy J    male  54.0      0
13                         Andersson, Mr. Anders Johan    male  39.0      1
18   Vander Planke, Mrs. Julius (Emelia Maria Vande…  female  31.0      1
..                                                 ...     ...   ...    ...
883                       Banfield, Mr. Frederick James    male  28.0      0
884                              Sutehall, Mr. Henry Jr    male  25.0      0
885                Rice, Mrs. William (Margaret Norton)  female  39.0      0
886                               Montvila, Rev. Juozas    male  27.0      0
890                                 Dooley, Mr. Patrick    male  32.0      0

     Parch            Ticket     Fare Cabin Embarked
0        0         A/5 21171   7.2500   NaN        S
4        0            373450   8.0500   NaN        S
6        0             17463  51.8625   E46        S
13       5            347082  31.2750   NaN        S
18       0            345763  18.0000   NaN        S
..     ...               ...      ...   ...      ...
883      0  C.A./SOTON 34068  10.5000   NaN        S
884      0    SOTON/OQ 392076   7.0500   NaN        S
885      5            382652  29.1250   NaN        Q
886      0            211536  13.0000   NaN        S
890      0            370376   7.7500   NaN        Q

[327 rows x 12 columns]
```

Q6: Display the top 10 rows of the 'Age' column with NAN value

[13]:
```python
print("URK21CS1128")
df6 = df[df.Age.isna()]
d6 = df6.head(10)
d6
```

```
URK21CS1128
```

[13]:
```
    PassengerId  Survived  Pclass  \
5             6         0       3
17           18         1       2
19           20         1       3
26           27         0       3
28           29         1       3
29           30         0       3
31           32         1       1
32           33         1       3
36           37         1       3
42           43         0       3

                                                 Name     Sex  Age  SibSp  Parch  \
5                                    Moran, Mr. James    male  NaN      0      0
17                       Williams, Mr. Charles Eugene    male  NaN      0      0
19                            Masselmani, Mrs. Fatima  female  NaN      0      0
26                            Emir, Mr. Farred Chehab    male  NaN      0      0
28                      O'Dwyer, Miss. Ellen "Nellie"  female  NaN      0      0
29                                Todoroff, Mr. Lalio    male  NaN      0      0
31  Spencer, Mrs. William Augustus (Marie Eugenie)  female  NaN      1      0
32                           Glynn, Miss. Mary Agatha  female  NaN      0      0
36                                   Mamee, Mr. Hanna    male  NaN      0      0
42                                Kraeff, Mr. Theodor    male  NaN      0      0

      Ticket      Fare Cabin Embarked
5     330877    8.4583   NaN        Q
17    244373   13.0000   NaN        S
19      2649    7.2250   NaN        C
26      2631    7.2250   NaN        C
28    330959    7.8792   NaN        Q
29    349216    7.8958   NaN        S
31  PC 17569  146.5208   B78        C
32    335677    7.7500   NaN        Q
36      2677    7.2292   NaN        C
42    349253    7.8958   NaN        C
```

Q7: Display the max value in ‘Fare’, min value in ‘Age’, and mean value in ‘Fare’

[14]:
```python
print("URK21CS1128")
df7=df['Fare'].max()
print(df7)
```

```
d7 = df['Age'].min()
print(d7)
d= df['Fare'].mean()
print(d)
```

```
URK21CS1128
512.3292
0.42
32.204207968574636
```

Q8: Display unique values in the Embarked column

[15]:
```
print("URK21CS1128")
df8 = df['Embarked'].unique()
df8
```

```
URK21CS1128
```

[15]:
```
array(['S', 'C', 'Q', nan], dtype=object)
```

Q9: Update the data frame with new column ‘New_Fare’. New_Fare = Fare + 100 and observe the size of the data frame

[16]:
```
df['New_Fare'] = df['Fare']+100
print(df)
df.shape[0]
```

```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           …         …       …
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                  Name     Sex   Age  SibSp  \
0                              Braund, Mr. Owen Harris    male  22.0      1
1    Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                               Heikkinen, Miss. Laina  female  26.0      0
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                             Allen, Mr. William Henry    male  35.0      0
..                                                 …     …   …    …
886                              Montvila, Rev. Juozas    male  27.0      0
887                       Graham, Miss. Margaret Edith  female  19.0      0
888           Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1
```

```
889                           Behr, Mr. Karl Howell    male  26.0      0
890                             Dooley, Mr. Patrick    male  32.0      0

     Parch            Ticket     Fare Cabin Embarked  New_Fare
0        0         A/5 21171   7.2500   NaN        S  107.2500
1        0          PC 17599  71.2833   C85        C  171.2833
2        0  STON/O2. 3101282   7.9250   NaN        S  107.9250
3        0            113803  53.1000  C123        S  153.1000
4        0            373450   8.0500   NaN        S  108.0500
..     …               …      …   …     …       …
886      0            211536  13.0000   NaN        S  113.0000
887      0            112053  30.0000   B42        S  130.0000
888      2        W./C. 6607  23.4500   NaN        S  123.4500
889      0            111369  30.0000  C148        C  130.0000
890      0            370376   7.7500   NaN        Q  107.7500

[891 rows x 13 columns]
```

[16]:

```
891
```

Q10: Drop the New_Fare column permanently and observe the size of the data frame

[17]:

```python
df10 = df.drop('New_Fare',axis=1,inplace=True)
print(df10)
df.shape[0]
```

```
None
```

[17]:

```
891
```

Q11: Drop the rows with NAN and observe the size of the data frame

[18]:

```python
df.dropna(inplace=True)
df
```

[18]:

```
     PassengerId  Survived  Pclass  \
1              2         1       1
3              4         1       1
6              7         0       1
10            11         1       3
11            12         1       1
..           …       …     …
871          872         1       1
872          873         0       1
879          880         1       1
887          888         1       1
889          890         1       1

                                                  Name     Sex   Age  SibSp  \
```

```
1    Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
6                                McCarthy, Mr. Timothy J    male  54.0      0
10                       Sandstrom, Miss. Marguerite Rut  female   4.0      1
11                              Bonnell, Miss. Elizabeth  female  58.0      0
..                                                   …     …   …     …
871    Beckwith, Mrs. Richard Leonard (Sallie Monypeny)  female  47.0      1
872                              Carlsson, Mr. Frans Olof    male  33.0      0
879       Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)  female  56.0      0
887                          Graham, Miss. Margaret Edith  female  19.0      0
889                                 Behr, Mr. Karl Howell    male  26.0      0

     Parch    Ticket     Fare        Cabin Embarked
1        0  PC 17599  71.2833          C85        C
3        0    113803  53.1000         C123        S
6        0     17463  51.8625          E46        S
10       1   PP 9549  16.7000           G6        S
11       0    113783  26.5500         C103        S
..     …       …      …       …        …
871      1     11751  52.5542          D35        S
872      0       695   5.0000  B51 B53 B55        S
879      1     11767  83.1583          C50        C
887      0    112053  30.0000          B42        S
889      0    111369  30.0000         C148        C

[183 rows x 12 columns]
```

Q12: Append two new rows in the data frame and observe the size of the data frame

[19]:
```python
new_rows = [{'Name': 'Bewin', 'Age':20, 'Sex': 'male', 'Survived': 1,
 'Embarked': 'S', 'Fare': 50},
            {'Name': 'John', 'Age': 25, 'Sex': 'male', 'Survived': 0,
 'Embarked': 'C', 'Fare': 80}]
df = df.append(new_rows, ignore_index=True)
print(df.shape)
```

```
(185, 12)

/tmp/ipykernel_3879147/563540454.py:3: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  df = df.append(new_rows, ignore_index=True)
```

Q13: Fill the NAN values in 'Cabin' column with 'A100' and observe the null values count

[20]:
```python
print("URK21CS1128")
d13=df['Cabin'].fillna('A100', inplace=True)
df13 = df['Cabin'].isnull().sum()
print(df13)
```

```
URK21CS1128
0
```

Q14: Group the rows based on the 'Embarked' column and observe how many are C = Cherbourg, Q = Queenstown, S = Southampton

[21]:
```python
grouped_embarked = df.groupby('Embarked').size()
print(grouped_embarked)
```

```
Embarked
C     66
Q      2
S    117
dtype: int64
```

Q15: Sort the data frame based on 'Fare'

[22]:
```python
df.sort_values(by='Fare', inplace=True)
print(df)
```

```
     PassengerId  Survived  Pclass  \
169        807.0         0     1.0
46         264.0         0     1.0
179        873.0         0     1.0
148        716.0         0     3.0
12          76.0         0     3.0
..           …         …     …
86         439.0         0     1.0
13          89.0         1     1.0
7           28.0         0     1.0
137        680.0         1     1.0
153        738.0         1     1.0

                                     Name     Sex   Age  SibSp  Parch  \
169                    Andrews, Mr. Thomas Jr    male  39.0    0.0    0.0
46                      Harrison, Mr. William    male  40.0    0.0    0.0
179                    Carlsson, Mr. Frans Olof    male  33.0    0.0    0.0
148  Soholt, Mr. Peter Andreas Lauritz Andersen    male  19.0    0.0    0.0
12                      Moen, Mr. Sigurd Hansen    male  25.0    0.0    0.0
..                                          …     …   …     …     …
86                          Fortune, Mr. Mark    male  64.0    1.0    4.0
13                   Fortune, Miss. Mabel Helen  female  23.0    3.0    2.0
7               Fortune, Mr. Charles Alexander    male  19.0    3.0    2.0
137          Cardeza, Mr. Thomas Drake Martinez    male  36.0    0.0    1.0
153                     Lesurer, Mr. Gustave J    male  35.0    0.0    0.0

      Ticket     Fare        Cabin Embarked
169   112050   0.0000          A36        S
46    112059   0.0000          B94        S
179      695   5.0000  B51 B53 B55        S
```

```
148    348124    7.6500        F G73        S
12     348123    7.6500        F G73        S
..        …        …           …       …
86       19950  263.0000  C23 C25 C27        S
13       19950  263.0000  C23 C25 C27        S
7        19950  263.0000  C23 C25 C27        S
137  PC 17755  512.3292  B51 B53 B55        C
153  PC 17755  512.3292         B101        C

[185 rows x 12 columns]
```

Result: The basic functionalities of python Data structures and data set using pandas were executed successfully.

[ ]:

[ ]:

Ex 3 Data Visualization through Python

September 4, 2023

Exp.No: 3

URK21CS1128

DATA VISUALIZATION THROUGH PYTHON

Aim: To execute the basic functionalities using data visualization with various charts.

Description:

Data visualization provides a good, organized pictorial representation of the data which makes it easier to understand, observe, analyze. In this tutorial, we will discuss how to visualize data using Python.Python provides various libraries that come with different features for visualizing data. All these libraries come with different features and can support various types of graphs. In this tutorial, we will be discussing four such libraries.

Matplotlib: Matplotlib is an easy-to-use, low-level data visualization library that is built on NumPy arrays. It consists of various plots like scatter plot, line plot, histogram, etc. Matplotlib provides a lot of flexibility.

Scatter Plot: Scatter plots are used to observe relationships between variables and uses dots to represent the relationship between them. The scatter() method in the matplotlib library is used to draw a scatter plot.

Line Chart: Line Chart is used to represent a relationship between two data X and Y on a different axis. It is plotted using the plot() function.

Bar Chart: A bar plot or bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent. It can be created using the bar() method.

Histogram: A histogram is basically used to represent data in the form of some groups. It is a type of bar plot where the X-axis represents the bin ranges while the Y-axis gives information about frequency. The hist() function is used to compute and create a histogram.

Program:

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
df = pd.read_csv("Emp_visu.csv")
print("URK21CS1128")
df
```

URK21CS1128

[1]:

	First Name	Gender	Salary	Bonus %	Senior Management
0	Maria	Female	130590	11.858	False
1	Angela	Female	54568	18.523	True
2	Allan	Male	125792	5.042	False
3	Rohan	Female	45906	11.598	True
4	Douglas	Male	97308	6.945	True
5	Brandon	Male	112807	17.492	True
6	Diana	Female	132940	19.082	False
7	Frances	Female	139852	7.524	True
8	Matthew	Male	100612	13.645	False
9	Larry	Male	101004	1.389	True
10	Joshua	Male	90816	18.816	True
11	Jerry	Male	72000	9.340	True
12	Lois	Female	64714	4.934	True
13	Dennis	Male	115163	10.125	False
14	John	Male	97950	13.873	False
15	Thomas	Male	61933	10.945	True
16	Shawn	Male	111737	6.414	False
17	Gary	Male	109831	5.831	False
18	Jeremy	Male	90370	7.369	False
19	Kimberly	Female	41426	7.450	True
20	Louise	Female	63241	15.132	True
21	Donna	Female	81014	1.894	False
22	Ruby	Female	65476	10.012	True
23	Lillian	Female	59414	1.256	False
24	Julie	Female	102508	12.637	True

\

	Team	Age	Experience	New_Salary	Incentive
0	Finance	26	5	146075.36220	20000
1	Business Development	27	5	64675.63064	19000
2	Client Services	28	6	132134.43260	18500
3	Finance	28	7	51230.17788	18000
4	Marketing	28	7	104066.04060	17000
5	Human Resources	30	8	132539.20040	16000
6	Client Services	31	9	158307.61080	15800
7	Business Development	34	10	150374.46450	15500
8	Marketing	34	10	114340.50740	15000
9	Client Services	35	11	102406.94560	14700
10	Client Services	35	11	107903.93860	14300
11	Finance	35	12	78724.80000	14000
12	Legal	35	12	67906.98876	14000
13	Legal	36	13	126823.25380	13000
14	Client Services	37	13	111538.60350	12000
15	Marketing	38	14	68711.56685	11900
16	Human Resources	39	15	118903.81120	11500

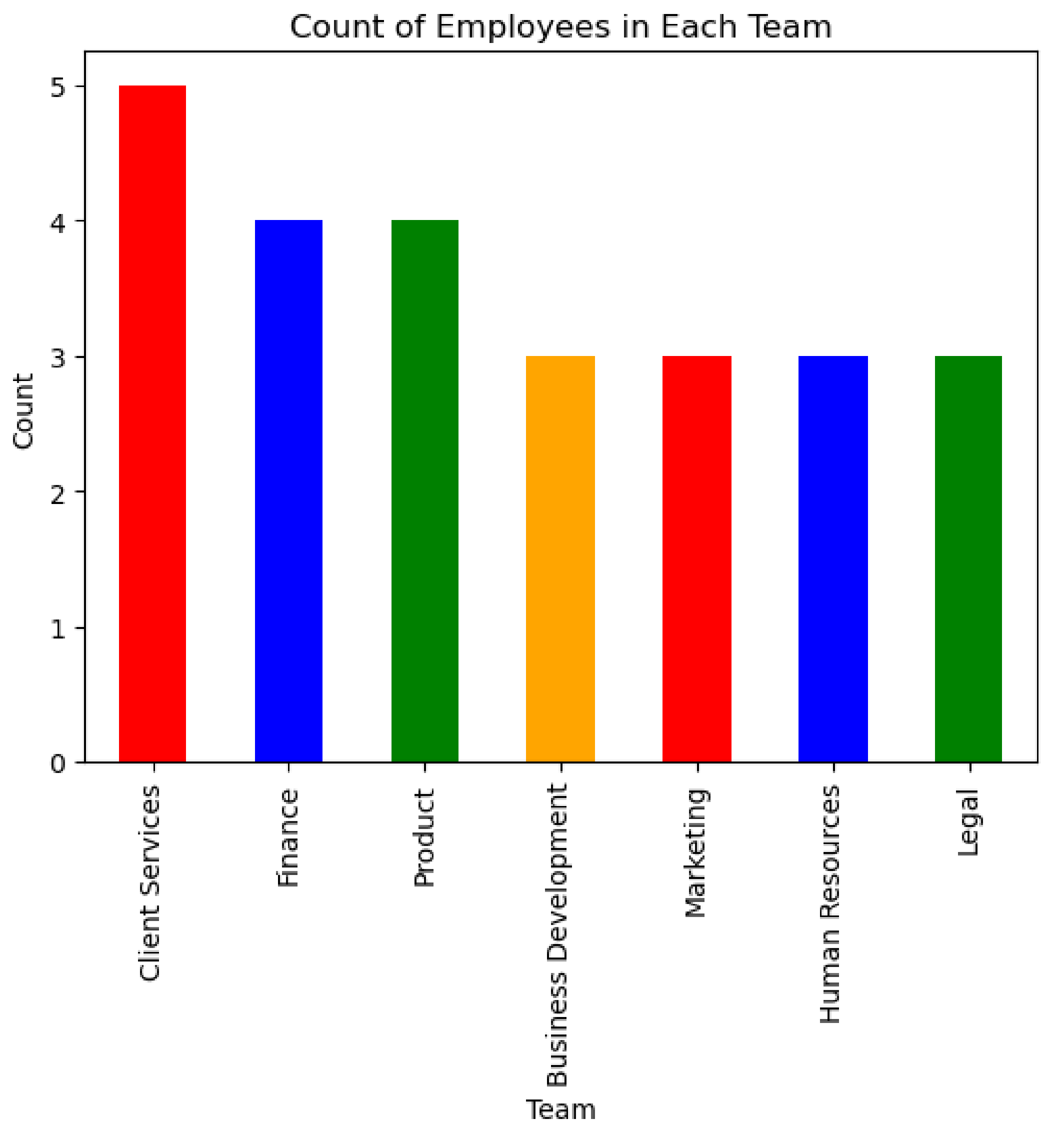
```
17               Product   39          15  116235.24560      11500
18       Human Resources   42          18   97029.36530      11000
19               Finance   44          20   44512.23700      11000
20  Business Development   45          21   72810.62812      10800
21               Product   49          23   82548.40516      10600
22               Product   54          25   72031.45712      10400
23               Product   55          26   60160.23984      10300
24                 Legal   58          27  115461.93600      10000
```

Q1: Draw a bar chart with Team and its count (use different colors for each team)

[4]:
```
print("URK21CS1128")
team_count = df['Team'].value_counts()
team_count.plot(kind='bar', color=['red', 'blue', 'green', 'orange'])
plt.xlabel('Team')
plt.ylabel('Count')
plt.title('Count of Employees in Each Team')
```

```
URK21CS1128
```

[4]:
```
Text(0.5, 1.0, 'Count of Employees in Each Team')
```

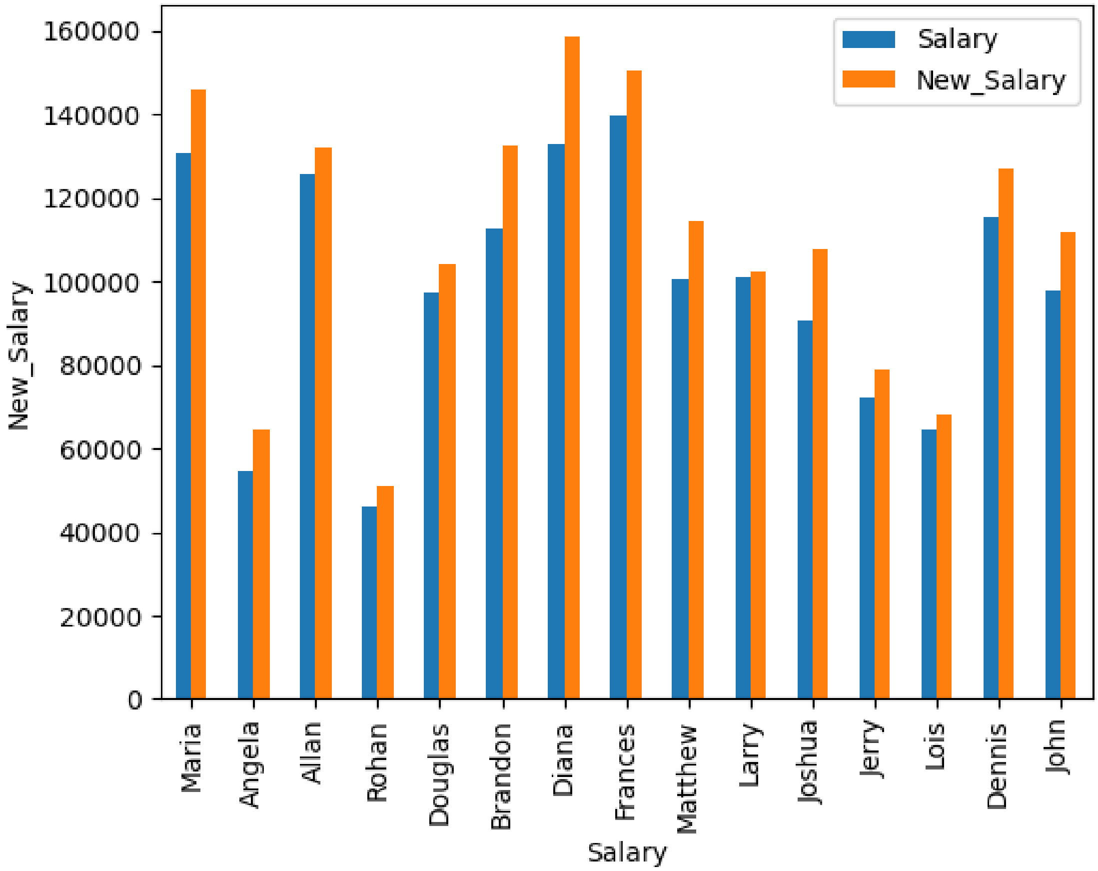


Q2: Draw a comparative bar chart for Salary and New_Salary against each person (first 15 persons)

[5]:
```
print("URK21CS1128")
df.head(15).plot(x='First Name', y=['Salary', 'New_Salary'], kind='bar')
plt.xlabel('Salary')
plt.ylabel('New_Salary')
```

URK21CS1128

[5]: Text(0, 0.5, 'New_Salary')

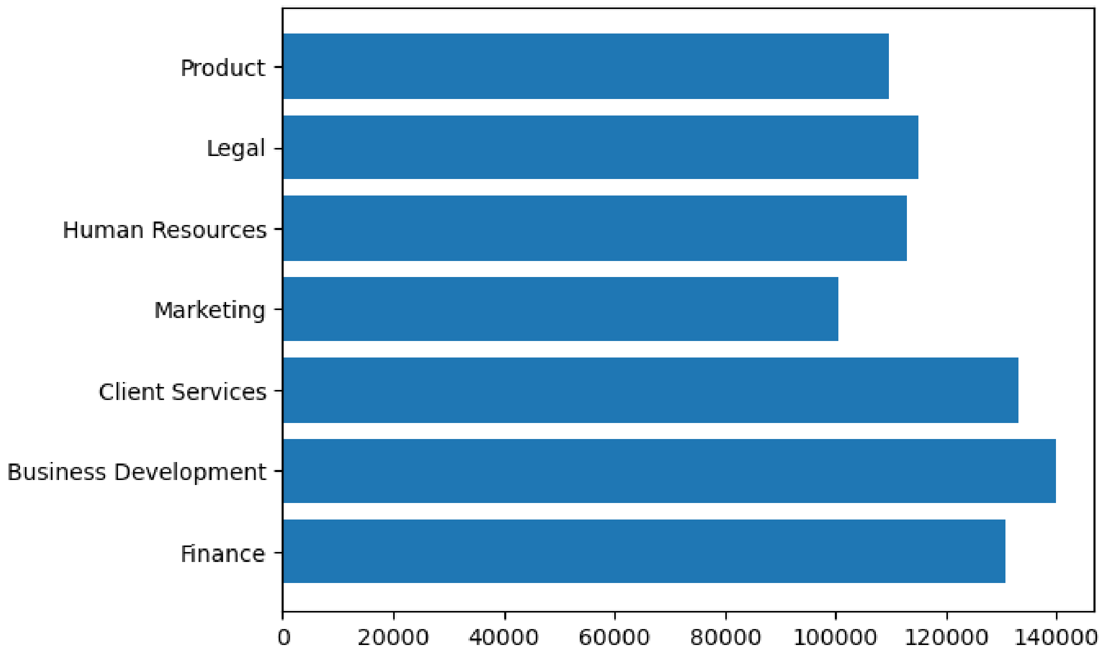


Q3: Draw a horizontal bar chart for Team and Salary

[6]:
```
print("URK21CS1128")
plt.barh(df["Team"],df["Salary"])
```

URK21CS1128

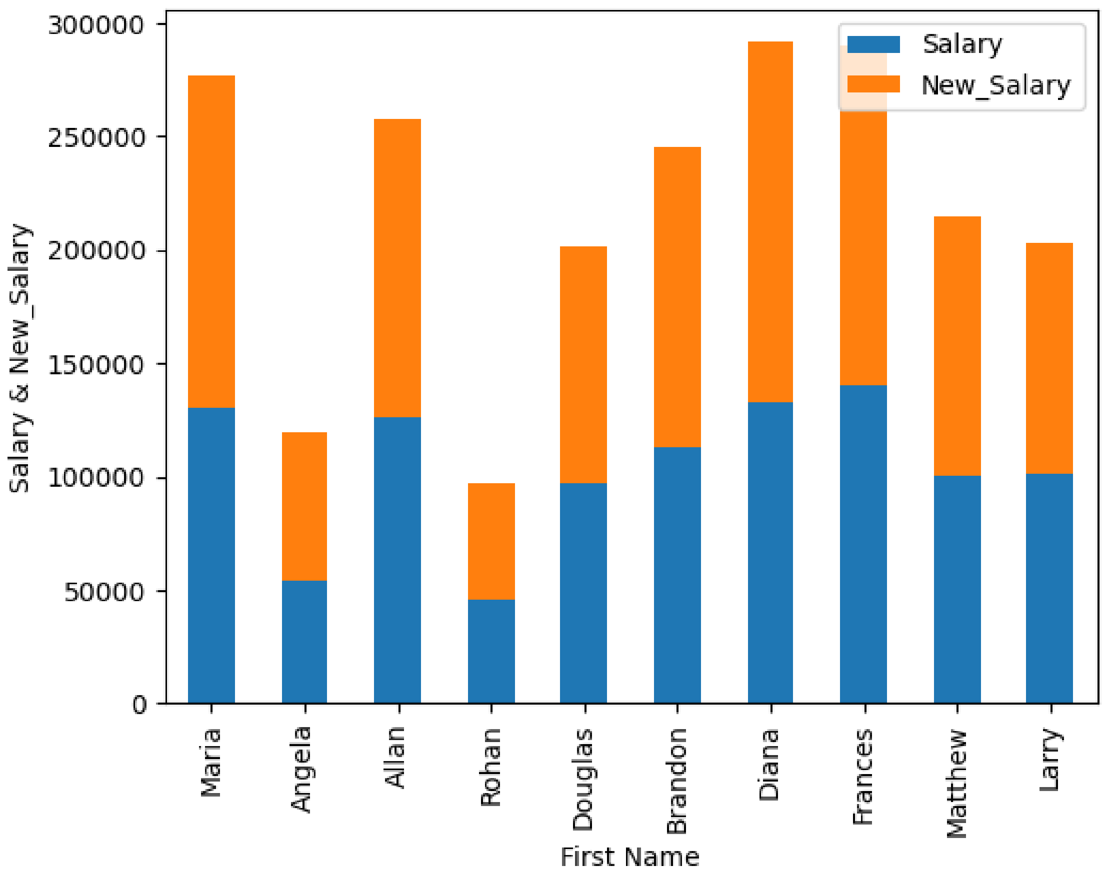
[6]: <BarContainer object of 25 artists>



Q4: Draw a stacked bar chart for Salary and New_price against the person (first 10 persons)

[7]:
```
print("URK21CS1128")
df.head(10).plot(x='First Name', y=['Salary','New_Salary'], kind='bar',␣
↪stacked=True)
plt.xlabel('First Name')
plt.ylabel('Salary & New_Salary')
plt.show()

# plt.figure(figsize = (8,5))
# plt.bar(df['First Name'].head(10),df['Salary'].head(10))
# plt.bar(df['First Name'].head(10),df['New_Salary'].
↪head(10),bottom=df['Salary'].head(10))
```

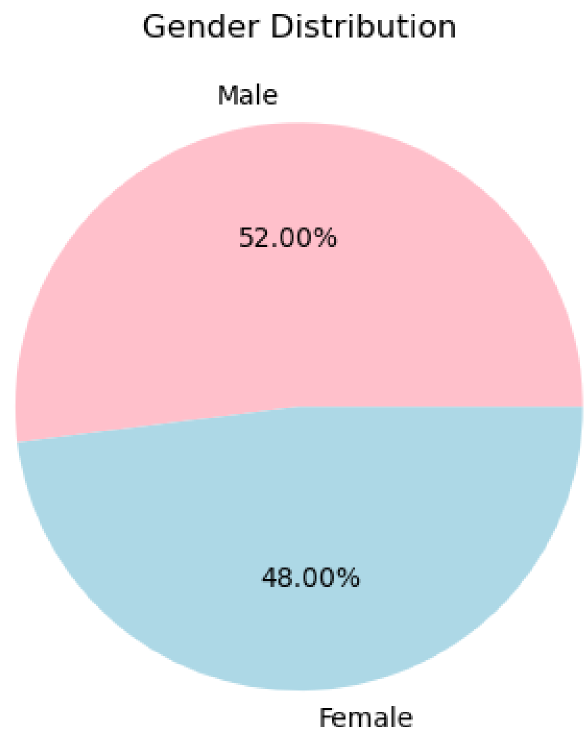
URK21CS1128



Q5: Draw a pie chart with Gender and its count

[8]:
```
print("URK21CS1128")
a = df['Gender'].value_counts()
plt.pie(df['Gender'].value_counts(), labels=a.index, autopct='%1.2f%%',␣
↪colors=['pink', 'lightblue'])
plt.title('Gender Distribution')
plt.show()
```

```
URK21CS1128
```

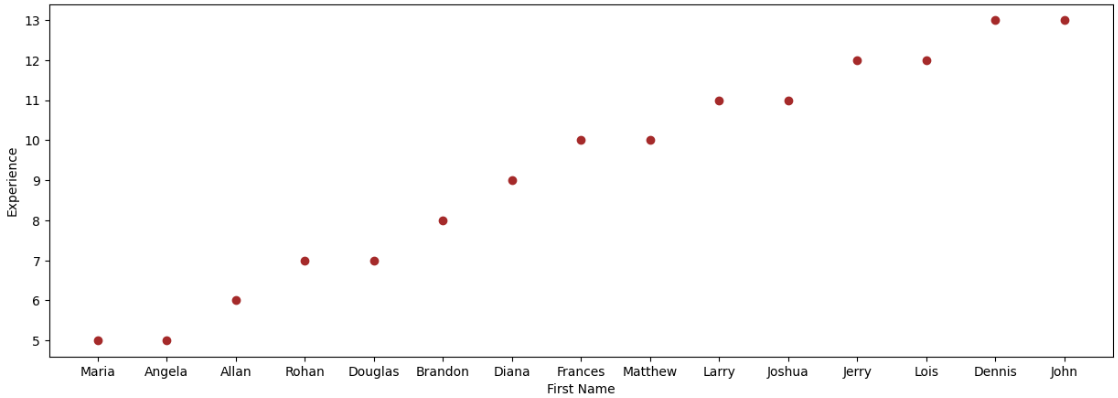


Q6: Draw the dot plot between person and experience (first 15 persons)

[9]:
```
print("URK21CS1128")
plt.figure(figsize = (15,5))
plt.plot(df["First Name"].head(15),df["Experience"].head(15),color =␣
↪"brown",marker = 'o',linewidth=0)
plt.xlabel("First Name")
plt.ylabel("Experience")
```

```
URK21CS1128
```

[9]:
```
Text(0, 0.5, 'Experience')
```

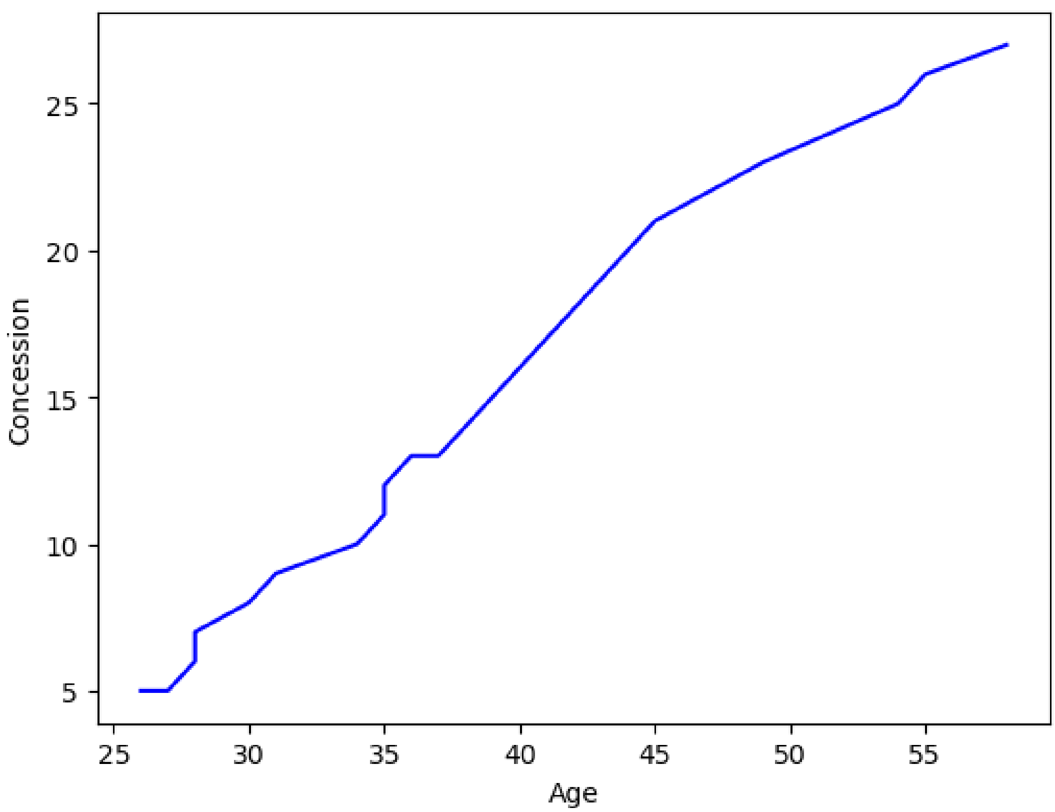


Q7: Draw the line plot between age and experience. Observe the trend line.

[10]:
```
print("URK21CS1128")
plt.plot(df["Age"],df["Experience"],color = "blue")
plt.xlabel("Age")
plt.ylabel("Concession")
```

```
URK21CS1128
```

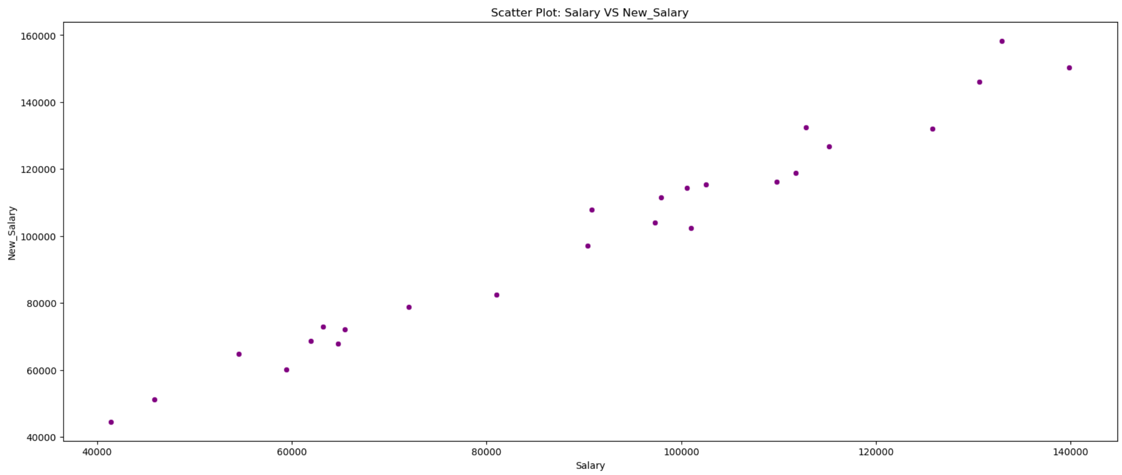
[10]:
```
Text(0, 0.5, 'Concession')
```



Q8: Draw the scatter plot between Salary and New_Salary. Observe the correlation

[11]:
```
print("URK21CS1128")
import seaborn as sns
plt.figure(figsize=(20,8))
sns.scatterplot(df, x='Salary', y='New_Salary', color='Purple')
plt.title('Scatter Plot: Salary VS New_Salary')
plt.xlabel('Salary')
plt.ylabel('New_Salary')
plt.show()
```

URK21CS1128

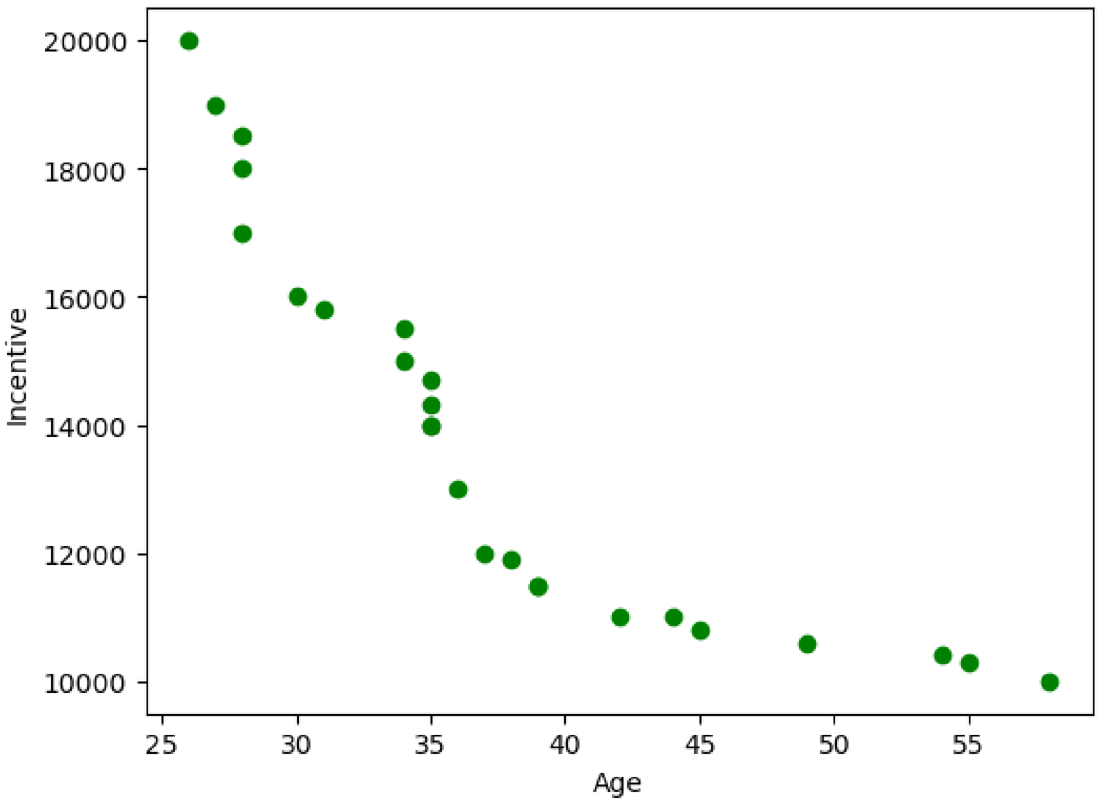


Q9: Draw the scatter plot between Age and Incentive. Observe the correlation

[12]:
```
print("URK21CS1128")
plt.scatter(df["Age"],df["Incentive"],color='green')
plt.xlabel("Age")
plt.ylabel("Incentive")
```

URK21CS1128

[12]: Text(0, 0.5, 'Incentive')



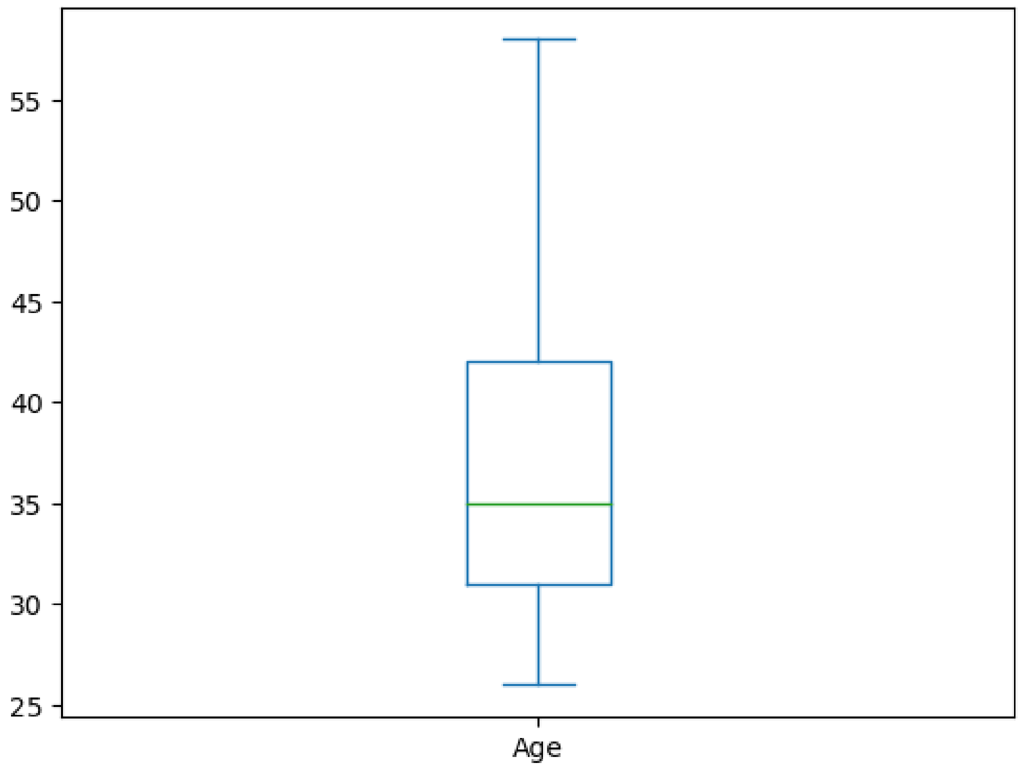
Q10: Draw the box plot to show the statistical summary of Age column

[13]:
```
print("URK21CS1128")
df["Age"].plot.box()
df.describe()
```

```
URK21CS1128
```

[13]:
```
              Salary   Bonus %        Age  Experience     New_Salary  \
count      25.000000  25.000000  25.000000   25.00000      25.000000
mean    90758.880000   9.965040  37.680000   13.72000   99898.113979
std     28441.424571   5.336828   8.938307    6.64906   32108.798871
min     41426.000000   1.256000  26.000000    5.00000   44512.237000
25%     64714.000000   6.414000  31.000000    9.00000   72031.457120
50%     97308.000000  10.012000  35.000000   12.00000  104066.040600
75%    111737.000000  13.645000  42.000000   18.00000  118903.811200
max    139852.000000  19.082000  58.000000   27.00000  158307.610800

          Incentive
count     25.000000
mean   13832.000000
std     3034.347816
```

```
min      10000.000000
25%      11000.000000
50%      14000.000000
75%      15800.000000
max      20000.000000
```

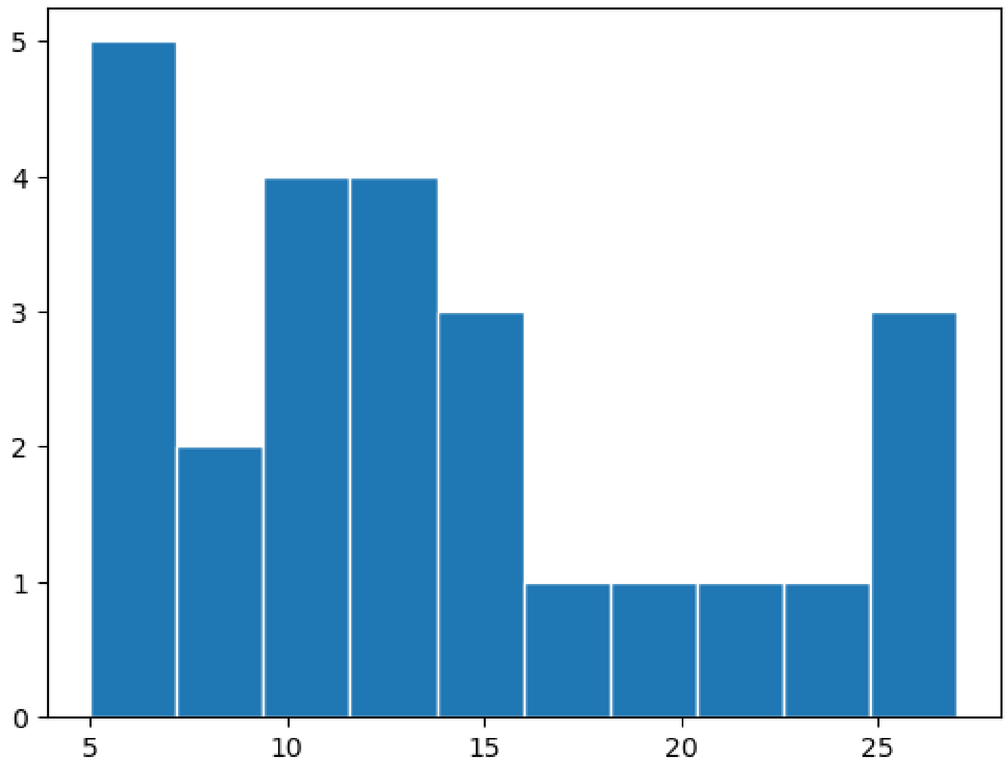


Q11: Draw the histogram plot for Experience column

[14]:
```
print("URK21CS1128")
plt.hist(df["Experience"], edgecolor='white')
```

```
URK21CS1128
```

[14]:
```
(array([5., 2., 4., 4., 3., 1., 1., 1., 1., 3.]),
 array([ 5. ,  7.2,  9.4, 11.6, 13.8, 16. , 18.2, 20.4, 22.6, 24.8, 27. ]),
 <BarContainer object of 10 artists>)
```

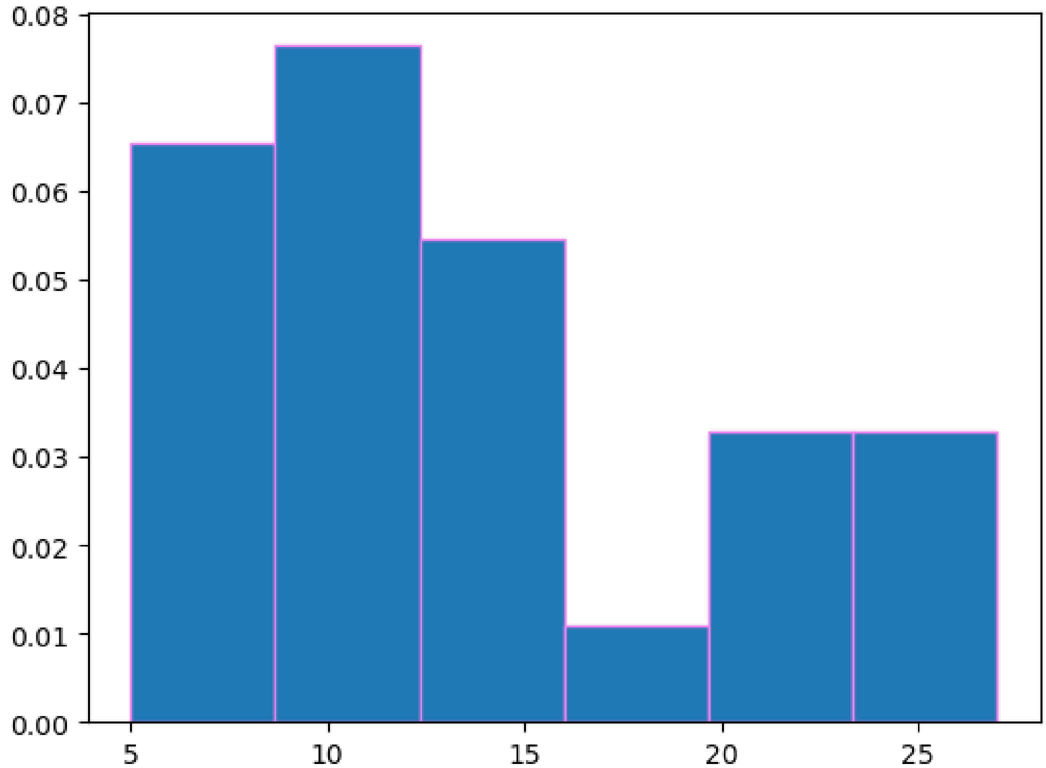


Q12: Draw the histogram plot for Experience column with bin value and PDF

[15]:
```
print("URK21CS1128")
plt.hist(df["Experience"],density='True', bins=6, edgecolor='violet')
```

```
URK21CS1128
```

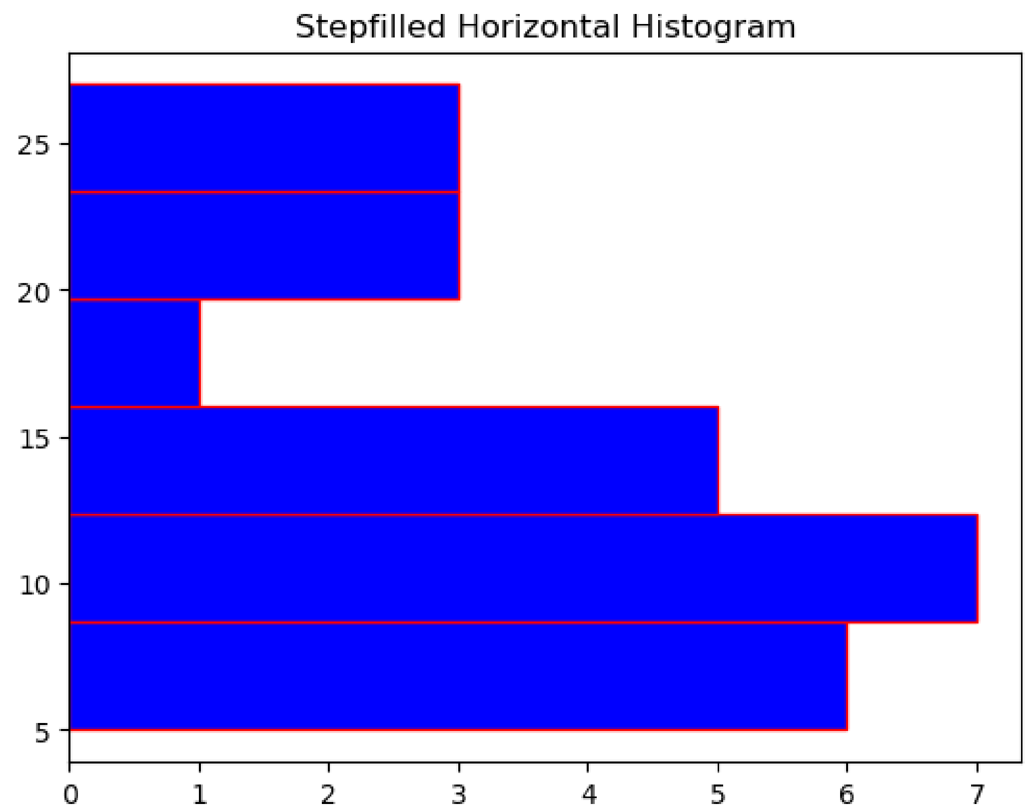
[15]:
```
(array([0.06545455, 0.07636364, 0.05454545, 0.01090909, 0.03272727,
        0.03272727]),
 array([ 5.        ,  8.66666667, 12.33333333, 16.        , 19.66666667,
        23.33333333, 27.        ]),
 <BarContainer object of 6 artists>)
```



13. Write code to change the horizontal histogram

[16]:
```
print("URK21CS1128")
plt.hist(df['Experience'], bins=6, orientation='horizontal', color='blue',␣
 ↪edgecolor='red')
plt.title('Stepfilled Horizontal Histogram')
plt.show()
```

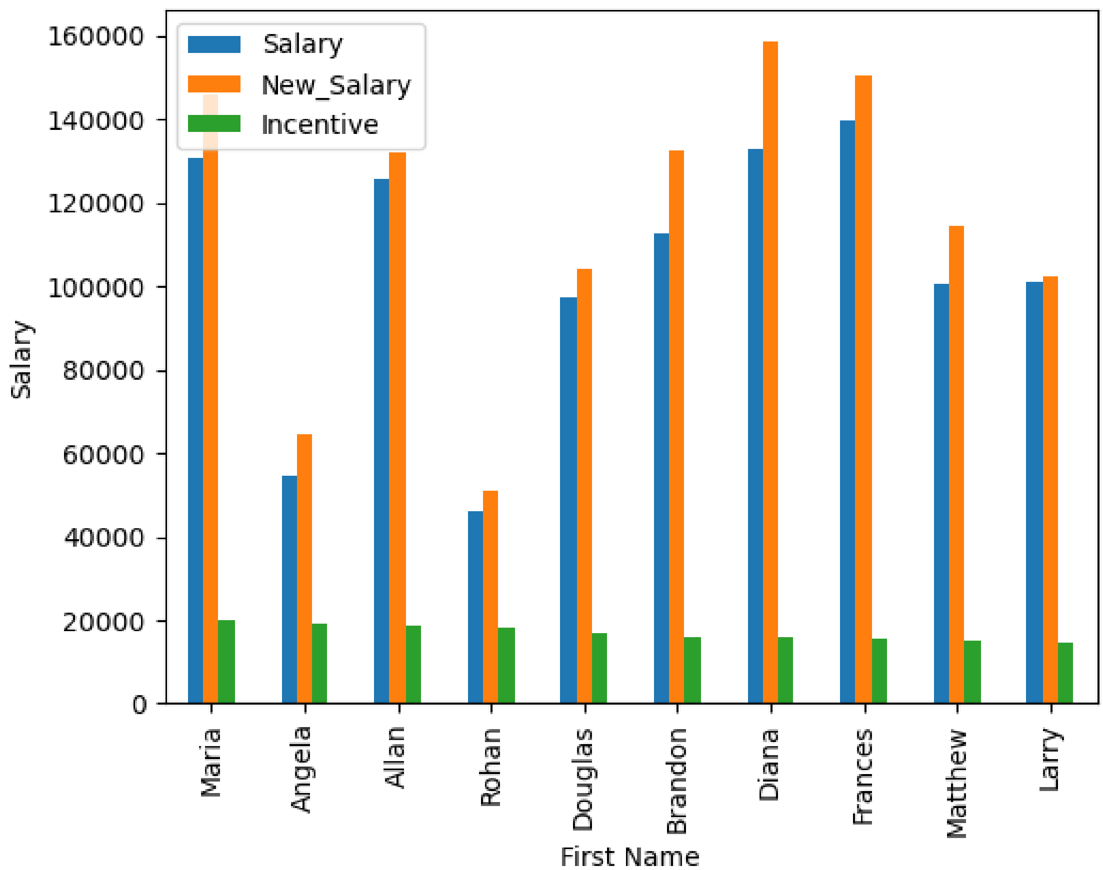
```
URK21CS1128
```



14. Compare any three features and display the comparative bar graph

[17]:
```
print("URK21CS1128")
comparison_data = df.head(10)[['First Name', 'Salary', 'New_Salary', 'Incentive']]
comparison_data.set_index('First Name').plot(kind='bar')
plt.xlabel('First Name')
plt.ylabel('Salary')
plt.show()
```

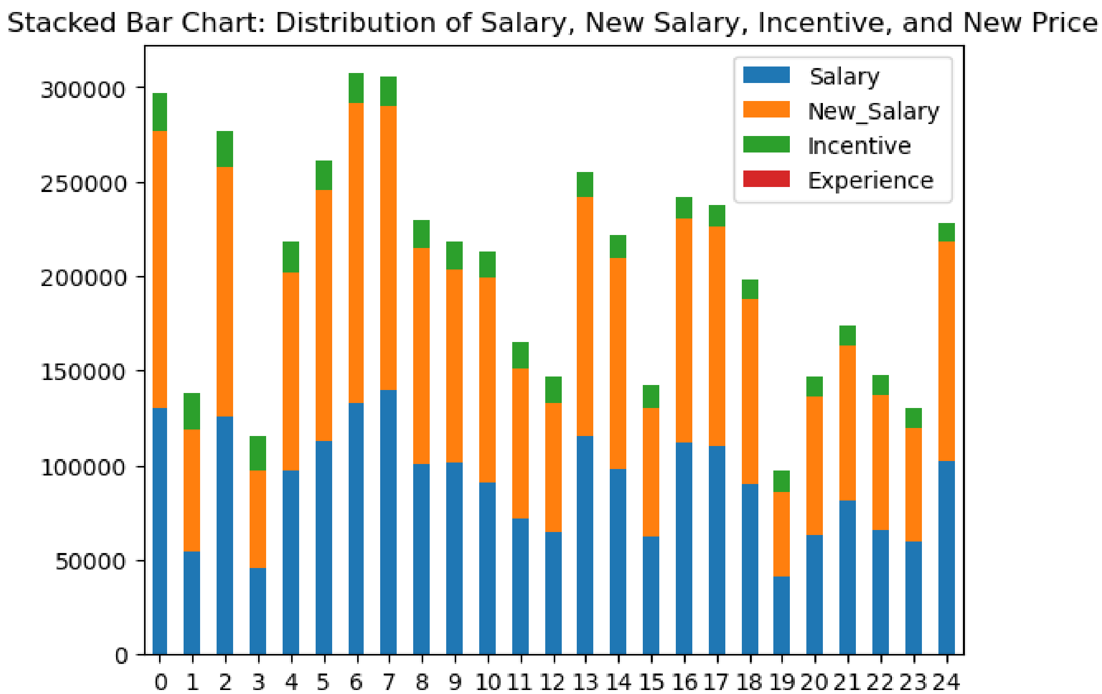
```
URK21CS1128
```



15. Stack any 4 features using a bar chart

[18]:
```
print("URK21CS1128")
df[['Salary', 'New_Salary', 'Incentive', 'Experience']].plot(kind='bar',␣
 ↪stacked=True)
plt.title('Stacked Bar Chart: Distribution of Salary, New Salary, Incentive,␣
 ↪and New Price')
plt.xticks(rotation=0)
plt.legend()
plt.show()
```

```
URK21CS1128
```



Result: The basic functionalities of data visualization using python were executed successfully.

Exp.4 Exploratory Data Analysis

September 4, 2023

URK21CS1128AIM: To perform exploratory data analysis on the given dataset using various python libraries. DESCRIPTION:

[1]:
```
import pandas as pd

df = pd.read_csv('iris_EDA.csv')
df
```

[1]:
```
    sepallength  sepalwidth  petallength  petalwidth            class Name  \
0           5.1         3.5          1.4         0.2      Iris-setosa   F1
1           4.9         3.0          1.4         0.2      Iris-setosa   F2
2           4.7         3.2          1.3         0.2      Iris-setosa   F3
3           4.6         3.1          1.5         0.2      Iris-setosa   F4
4           5.0         3.6          1.4         0.2      Iris-setosa   F5
5           5.4         3.9          1.7         0.4      Iris-setosa   F6
6           4.6         3.4          1.4         0.3      Iris-setosa   F7
7           5.0         3.4          1.5         0.2      Iris-setosa   F8
8           7.0         3.2          4.7         1.4  Iris-versicolor   F9
9           6.4         3.2          4.5         1.5  Iris-versicolor  F10
10          6.9         3.1          4.9         1.5  Iris-versicolor  F11
11          5.5         2.3          4.0         1.3  Iris-versicolor  F12
12          6.5         2.8          4.6         1.5  Iris-versicolor  F13
13          5.7         2.8          4.5         1.3  Iris-versicolor  F14
14          6.3         3.3          4.7         1.6  Iris-versicolor  F15
15          4.9         2.4          3.3         1.0  Iris-versicolor  F16
16          6.3         3.3          6.0         2.5   Iris-virginica  F17
17          5.8         2.7          5.1         1.9   Iris-virginica  F18
18          7.1         3.0          5.9         2.1   Iris-virginica  F19
19          6.3         2.9          5.6         1.8   Iris-virginica  F20
20          6.5         3.0          5.8         2.2   Iris-virginica  F21
21          7.6         3.0          6.6         2.1   Iris-virginica  F22
22          4.9         2.5          4.5         NaN   Iris-virginica  F23
23          7.3         2.9          6.3         1.8   Iris-virginica  F24
24          7.3         2.9          6.3         1.8   Iris-virginica  F24

    Score   Color
0    12.0     Red
1     NaN    Blue
```

```
2     18.0  Orange
3     14.0  Purple
4     22.0     Red
5     27.0    Blue
6     24.0  Orange
7     23.0  Purple
8     16.0     Red
9     19.0    Blue
10    21.0  Orange
11    25.0  Purple
12    28.0     Red
13    29.0    Blue
14    11.0  Orange
15    30.0  Purple
16    12.0     Red
17    24.0    Blue
18    17.0  Orange
19    15.0  Purple
20    22.0     Red
21    27.0    Blue
22    25.0  Orange
23    21.0  Purple
24    21.0  Purple
```

Q1: Remove the irrelevant column 'Color' and display top 5 rows (use inplace=True)

[2]:
```
print(1128)

df.drop('Color',axis=1,inplace=True)
print('Column dropped from dataframe permanently.')
```

```
1128
Column dropped from dataframe permanently.
```

[3]:
```
print(1128)
df.shape
```

```
1128
```

[3]:
```
(25, 7)
```

Q2: Remove the duplicate rows and display the shape of the dataframe(use inplace=True).

[4]:
```
print(1128)

df.drop_duplicates(keep='first',inplace=True)  #use 'subset' attribute for␣
↪dropping duplicates in individual columns
print('Dropped the duplicate rows.')
df.shape
```

```
1128
Dropped the duplicate rows.
```

[4]:
```
(24, 7)
```

Q3: Rename the column 'class' to 'Category' and display top 5 rows (use inplace=True).

[5]:
```python
print(1128)
df.rename(columns={'class':'Category'},inplace=True)
print("Changed the column name 'class' to 'category' in the dataframe.")
df.head()
```

```
1128
Changed the column name 'class' to 'category' in the dataframe.
```

[5]:

	sepallength	sepalwidth	petallength	petalwidth	Category	Name	Score
0	5.1	3.5	1.4	0.2	Iris-setosa	F1	12.0
1	4.9	3.0	1.4	0.2	Iris-setosa	F2	NaN
2	4.7	3.2	1.3	0.2	Iris-setosa	F3	18.0
3	4.6	3.1	1.5	0.2	Iris-setosa	F4	14.0
4	5.0	3.6	1.4	0.2	Iris-setosa	F5	22.0

Q4:Drop the missing value row-wise and display the shape of dataframe (use inplace=False).

[6]:
```python
print(1128)
df.dropna(axis=0,inplace=True)
print('Dropped the rows with null/missing values in the dataframe.')
df.shape
```

```
1128
Dropped the rows with null/missing values in the dataframe.
```

[6]:
```
(22, 7)
```

Q5:Calculate the central tendency measures for 'Score' and display the same.

[7]:
```python
print(1128)

print('Mean: ', df['Score'].mean())
print('Median: ', df['Score'].median())
print('Mode: ', df['Score'].mode())
```

```
1128
Mean:  20.772727272727273
Median:  21.5
Mode:  0    12.0
1    21.0
2    22.0
3    24.0
4    27.0
Name: Score, dtype: float64
```

Q6.Calculate the variability measures for 'Score' and display the same.

[8]:
```
print(1128)
x = df['Score'].min()
y = df['Score'].max()
print('Variability Measures for the column-Score: ')
print('Max: ',y)
print('Min: ',x)
print('Range:',(y-x))
print('Standard Deviation: ',df['Score'].std())
print('Variance: ',df['Score'].var())
```

```
1128
Variability Measures for the column-Score:
Max:  30.0
Min:  11.0
Range: 19.0
Standard Deviation:  5.797633492430693
Variance:  33.612554112554115
```

Q7.Calculate the IQR using quantile for 'Score' and display the same.

[9]:
```
print(1128)
Q1 = df['Score'].quantile(.25)
Q3 = df['Score'].quantile(.75)
print('IQR: ',(Q3-Q1))  #IQR formula=Q3-Q1
```

```
1128
IQR:  8.5
```

Q8. Calculate the z-score for 'Score' and display the same.

[10]:
```
print(1128)
#z-score = x-mean/SD
import  scipy.stats as stats

zscore = stats.zscore(df['Score'])
print('Z-score:',zscore)
```

```
1128
Z-score: 0    -1.548765
2    -0.489506
3    -1.195679
4     0.216667
5     1.099383
6     0.569753
7     0.393210
8    -0.842592
9    -0.312963
10    0.040123
```

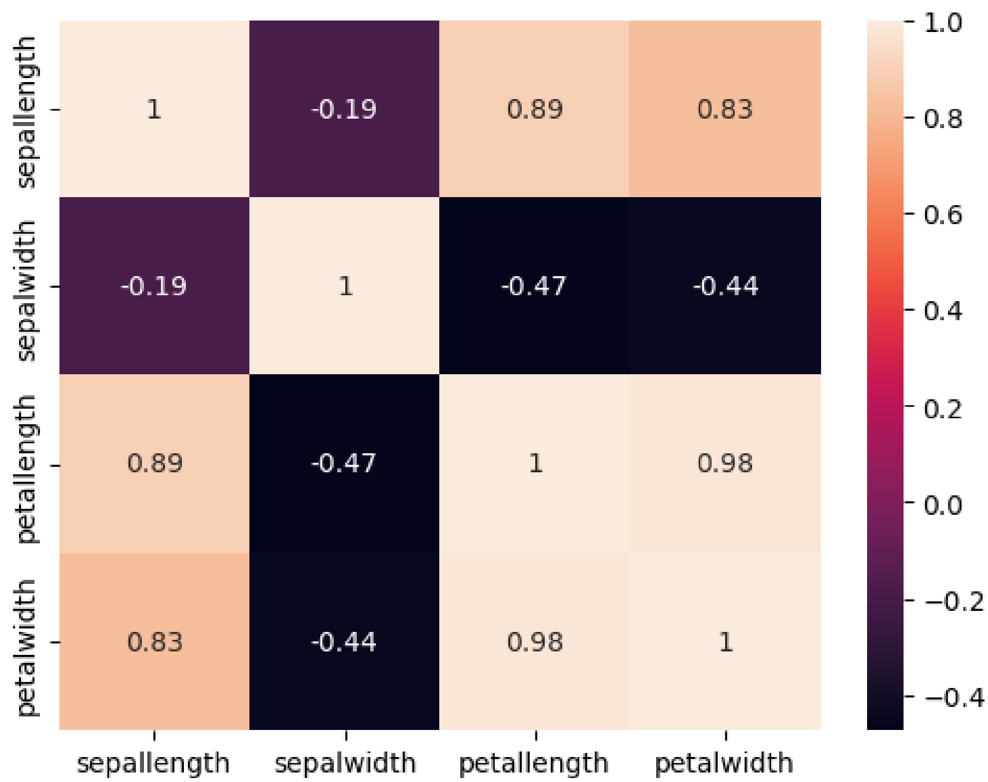
```
11    0.746296
12    1.275926
13    1.452469
14   -1.725308
15    1.629012
16   -1.548765
17    0.569753
18   -0.666049
19   -1.019136
20    0.216667
21    1.099383
23    0.040123
Name: Score, dtype: float64
```

Q9: Plot the heatmap using the correlation ('sepallength', 'sepalwidth', 'petallength', 'petalwidth').

[11]:
```python
print(1128)
import seaborn as sns

t = df[['sepallength', 'sepalwidth', 'petallength', 'petalwidth']]
c = t.corr()
sns.heatmap(c, xticklabels = c.columns, yticklabels = c.columns, annot = True)
```

```
1128
```

[11]:
```
<Axes: >
```



Q10:Add 2 rows at the end of the dataframe with the given values and display last 5 rows

{'sepallength':7.6,'sepalwidth':2.9,'petallength':5.3,'petalwidth':2.1,'Category':'Iris-virginica','Name':'F25','Score':80}

df2={'sepallength':4.6,'sepalwidth':1.3,'petallength':0.3,'Category':'Iris-setosa','Name':'F26','Score':85}

[19]:
```
print(1128)

df1 = {'sepallength':7.6,'sepalwidth':2.9,'petallength':5.3,'petalwidth':2.
↪1,'Category':'Iris-virginica','Name':'F25','Score':80}

df2 = {'sepallength':4.6,'sepalwidth':1.3,'petallength':0.3,'Category':
↪'Iris-setosa','Name':'F26','Score':85}

df = df.append(df1,ignore_index=True)
df = df.append(df2,ignore_index=True)
df.tail()
```

```
1128

/tmp/ipykernel_3676372/3128689531.py:7: FutureWarning: The frame.append method
is deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  df = df.append(df1,ignore_index=True)
/tmp/ipykernel_3676372/3128689531.py:8: FutureWarning: The frame.append method
is deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  df = df.append(df2,ignore_index=True)
```

[19]:
```
    sepallength  sepalwidth  petallength  petalwidth         Category Name  \
23          4.6         1.3          0.3    1.273913      Iris-setosa  F26
24          7.6         2.9          5.3    2.100000  Iris-virginica  F25
25          4.6         1.3          0.3         NaN      Iris-setosa  F26
26          7.6         2.9          5.3    2.100000  Iris-virginica  F25
27          4.6         1.3          0.3         NaN      Iris-setosa  F26

    Score
23   85.0
24   80.0
25   85.0
26   80.0
27   85.0
```

Q11: Replace NaN value in 'petalwidth' with mean petalwidth values and display last 5 rows.

[13]:
```python
print(1128)
import numpy as np
m= df['petalwidth'].mean()
df.replace(to_replace=np.nan, value=m, inplace=True)
df.tail()
```

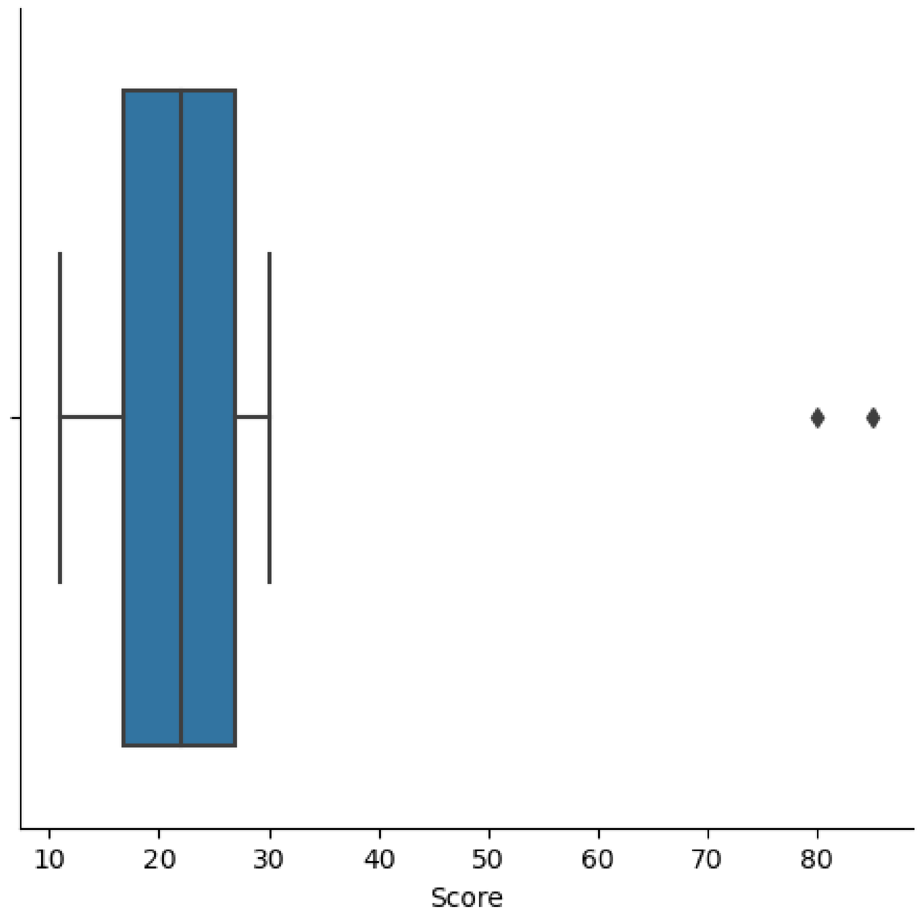
```
1128
```

[13]:
```
    sepallength  sepalwidth  petallength  petalwidth        Category Name  \
19          6.5         3.0          5.8    2.200000  Iris-virginica  F21
20          7.6         3.0          6.6    2.100000  Iris-virginica  F22
21          7.3         2.9          6.3    1.800000  Iris-virginica  F24
22          7.6         2.9          5.3    2.100000  Iris-virginica  F25
23          4.6         1.3          0.3    1.273913     Iris-setosa  F26

    Score
19   22.0
20   27.0
21   21.0
22   80.0
23   85.0
```

Q12:Detect the outliers in 'Score' with boxplot.

[14]:
```python
print(1128)
sns.catplot(x='Score', kind='box', data=df)
print('Mean: ', df['Score'].mean())
print('Standard Deviation: ',df['Score'].std())
print('Variance: ',df['Score'].var())
```

```
1128
Mean:  25.916666666666668
Standard Deviation:  18.301619473760205
Variance:  334.9492753623188
```

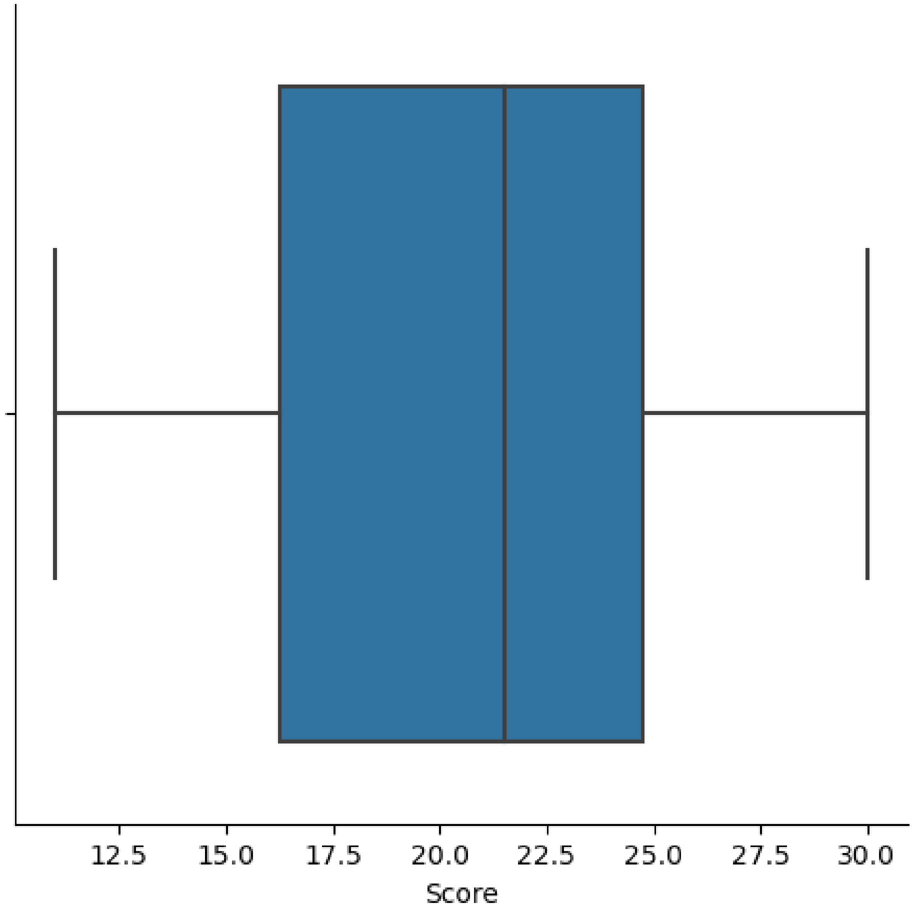


Q13:Remove the outliers using IQR and recalculate IQR in outlier removed ‘Score’ column and analyse with boxplot (Use df.copy()).

[15]:
```
print(1128)
Q1 = df['Score'].quantile(.25)
Q3 = df['Score'].quantile(.75)
IQR = Q3-Q1
print(Q1,Q3)
print('IQR: ',(Q3-Q1))
l = Q1-1.5*IQR
h = Q3+1.5*IQR
new_frame = df[(df['Score']>l) & (df['Score']<h)]
new_frame.shape
new_frame.tail()
sns.catplot(x='Score', kind='box', data=new_frame)
```

```
1128
16.75 27.0
IQR:  10.25
```

[15]: <seaborn.axisgrid.FacetGrid at 0x7f89126b5f40>



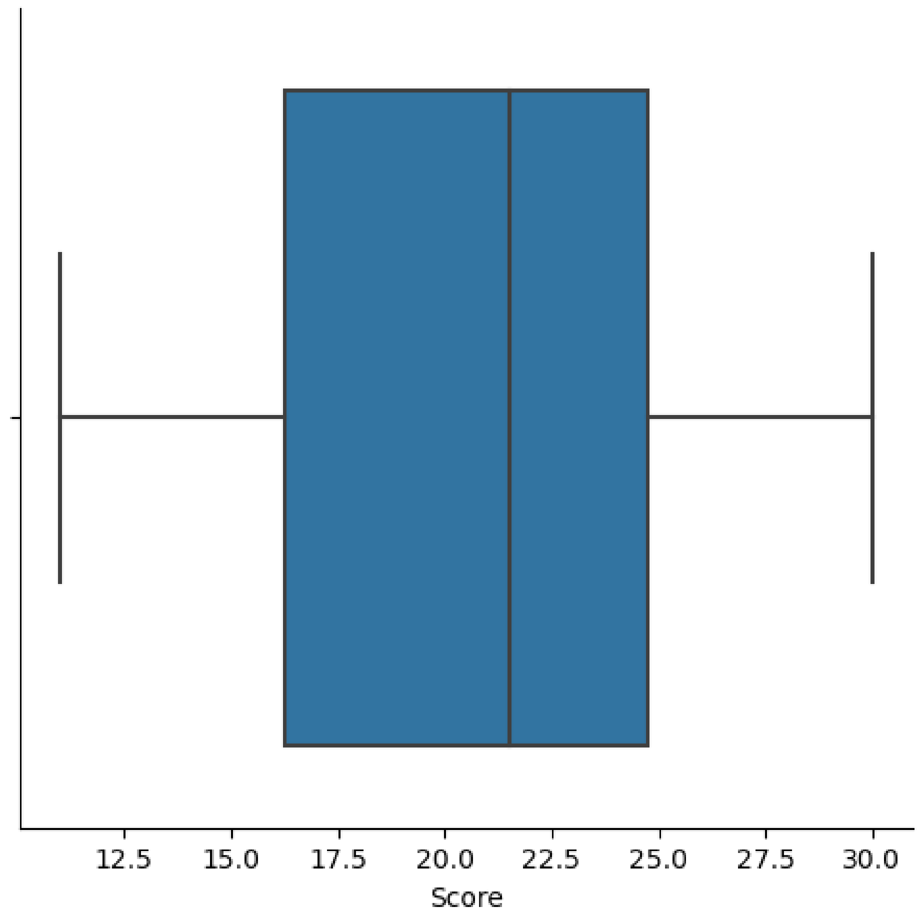
Q14: Remove the outliers using z-score and recalculate z-score in outlier removed 'Score' and analyse with boxplot (Use df.copy()).

[16]:
```
print(1128)
zscore = stats.zscore(df['Score'])
print('Z-score:',zscore)

filtered = (zscore<3)
new_df2 = df[filtered]
new_df2.tail()
sns.catplot(x='Score', kind='box', data=new_df2)
```

```
1128
Z-score: 0    -0.776761
1    -0.441870
2    -0.665131
3    -0.218609
```

```
4     0.060466
5    -0.106979
6    -0.162794
7    -0.553500
8    -0.386055
9    -0.274425
10   -0.051164
11    0.116282
12    0.172097
13   -0.832576
14    0.227912
15   -0.776761
16   -0.106979
17   -0.497685
18   -0.609316
19   -0.218609
20    0.060466
21   -0.274425
22    3.018670
23    3.297746
Name: Score, dtype: float64
```

```
[16]: <seaborn.axisgrid.FacetGrid at 0x7f89126c7a60>
```



Q15:Drop the last two rows added in the dataframe.

```
[2]: #15 Drop the last two rows added in the dataframe
print('URK21CS1128')
df = df.drop([22,23])
df.shape
print(df.to_string())
```

```
URK21CS1128
    sepallength  sepalwidth  petallength  petalwidth            class Name
Score   Color
0           5.1         3.5          1.4         0.2      Iris-setosa   F1
12.0     Red
1           4.9         3.0          1.4         0.2      Iris-setosa   F2
NaN    Blue
2           4.7         3.2          1.3         0.2      Iris-setosa   F3
18.0  Orange
3           4.6         3.1          1.5         0.2      Iris-setosa   F4
14.0  Purple
```

```
4           5.0         3.6         1.4         0.2       Iris-setosa   F5
22.0     Red
5           5.4         3.9         1.7         0.4       Iris-setosa   F6
27.0    Blue
6           4.6         3.4         1.4         0.3       Iris-setosa   F7
24.0  Orange
7           5.0         3.4         1.5         0.2       Iris-setosa   F8
23.0  Purple
8           7.0         3.2         4.7         1.4   Iris-versicolor   F9
16.0     Red
9           6.4         3.2         4.5         1.5   Iris-versicolor  F10
19.0    Blue
10          6.9         3.1         4.9         1.5   Iris-versicolor  F11
21.0  Orange
11          5.5         2.3         4.0         1.3   Iris-versicolor  F12
25.0  Purple
12          6.5         2.8         4.6         1.5   Iris-versicolor  F13
28.0     Red
13          5.7         2.8         4.5         1.3   Iris-versicolor  F14
29.0    Blue
14          6.3         3.3         4.7         1.6   Iris-versicolor  F15
11.0  Orange
15          4.9         2.4         3.3         1.0   Iris-versicolor  F16
30.0  Purple
16          6.3         3.3         6.0         2.5    Iris-virginica  F17
12.0     Red
17          5.8         2.7         5.1         1.9    Iris-virginica  F18
24.0    Blue
18          7.1         3.0         5.9         2.1    Iris-virginica  F19
17.0  Orange
19          6.3         2.9         5.6         1.8    Iris-virginica  F20
15.0  Purple
20          6.5         3.0         5.8         2.2    Iris-virginica  F21
22.0     Red
21          7.6         3.0         6.6         2.1    Iris-virginica  F22
27.0    Blue
24          7.3         2.9         6.3         1.8    Iris-virginica  F24
21.0  Purple
```

Result:

[ ]:
```
The basic functionalities of data visualization using python were executed
  successfully.
```

[ ]:
```
```

[ ]:
```
```

EXP05 - Statistical Inference

September 4, 2023

Exp_No: 05

Reg No.: URK21CS1128

Bewin Felix R A

Aim:

To demonstrate the statistical interferences used for data science application using python language.

Description:

Inferential statistics are used to draw inferences from the sample of a huge data set. Random samples of data are taken from a population, which are then used to describe and make inferences and predictions about the population.

Sample Mean and Population Mean:

Sample mean is the arithmetic mean of random sample values drawn from the population. Population mean represents the actual mean of the whole population. If the sample is random and sample size is large then the sample mean would be a good estimate of the population mean.

Correlation Coefficient:

The correlation coefficient quantifies the relationship between the two variables. There are two methods of calculating the Correlation Coefficient and its matrix – Pearson and Spearman.

Covariance Matrix:

It is a square matrix giving the covariance between each pair of elements of a given random vector.

Hypothesis Testing using Z Test:

Hypothesis testing is a statistical method that is used in making statistical decisions using experimental data. One of the ways to perform hypothesis testing is Z-test, where the Two-sample Z-test is used to test whether the two datasets are similar or not. Also, Z-test is used when the sample size is greater than 30.

Confidence Interval:

A confidence interval displays the probability that a parameter will fall between a pair of values around the mean. Confidence intervals measure the degree of uncertainty or certainty in a sampling method. They are most often constructed using confidence levels of 95% or 99%.

[13]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

```
import numpy as np
import scipy.stats as stats
import math
```

[34]:
```
print('URK21CS1128')
path = "supermarket.csv"
df = pd.read_csv(path) #read the csv file
print(df.shape) #3000 - population
```

```
URK21CS1128
(3000, 17)
```

[15]:
```
# Lets take seed so that everytime the random values come out to be constant
np.random.seed(6)
```

[16]:
```
#1. Calculate the sample mean for 'Unit price' column with n=500 and observe
print('URK21CS1128')
s1 = 500
sample_1 = np.random.choice(a = df['Unit price'] , size = s1)
m_sample_1 = sample_1.mean();
print("The Mean of the sample data of Unit price for 500 :", m_sample_1)
```

```
URK21CS1128
The Mean of the sample data of Unit price for 500 : 55.11994
```

[17]:
```
#2. Calculate the sample mean for 'Unit price' column with n=1000 and observe
print('URK21CS1128')
s2 = 1000

sample_2 = np.random.choice(a = df['Unit price'] , size = s2)
m_sample_2 = sample_2.mean();
print("The Mean of the sample data of Unit price for 1000 :", m_sample_2 )
```

```
URK21CS1128
The Mean of the sample data of Unit price for 1000 : 56.03247999999999
```

[18]:
```
#3. Calculate the population mean for 'Unit price' column
print('URK21CS1128')
pm = df['Unit price'].mean()
print("The Mean of the population data of Unit price :", pm)
```

```
URK21CS1128
The Mean of the population data of Unit price : 55.67213
```

[19]:
```
#4. Calculate the confidence interval (CI) with sample mean for 'Unit price'␣
↪column of n=500 and confidence level of 95%. Observe whether the population␣
↪mean lies in CI.
print('URK21CS1128')
print( "Sample mean of 500 samples:", m_sample_1)
```

```
SD = sample_1.std()
print("Sample SD of 500 samples:", SD)

CL=0.95
alpha=(1-CL)/2
z_critical = round(stats.norm.ppf(1-alpha),2)
print("Z-score:", z_critical)

er=z_critical*(SD/math.sqrt(s1))
L=m_sample_1-er
H=m_sample_1+er

print("Confidence Level", L, H)
print("[",L,pm,H,"]")
```

```
URK21CS1128
Sample mean of 500 samples: 55.11994
Sample SD of 500 samples: 26.150292078605926
Z-score: 1.96
Confidence Level 52.827765835805906 57.412114164194094
[ 52.827765835805906 55.67213 57.412114164194094 ]
```

[20]:
```
#5. Change the confidence level to 99% and observe the confidence interval for␣
↪the same sample mean for 'Unit price' column of n=500.
print('URK21CS1128')
print( "Sample mean of 500 samples:", m_sample_1)

SD = sample_1.std()
print("Sample SD of 500 samples:", SD)

CL=0.9
alpha=(1-CL)/2
z_critical = round(stats.norm.ppf(1-alpha),2)
print("Z-score:", z_critical)

er=z_critical*(SD/math.sqrt(s1))
L=m_sample_1-er
H=m_sample_1+er

print("Confidence Level", L, H)
print("[",L,pm,H,"]")
```

```
URK21CS1128
Sample mean of 500 samples: 55.11994
Sample SD of 500 samples: 26.150292078605926
Z-score: 1.64
Confidence Level 53.20199835240902 57.03788164759098
```

```
[ 53.20199835240902 55.67213 57.03788164759098 ]
```

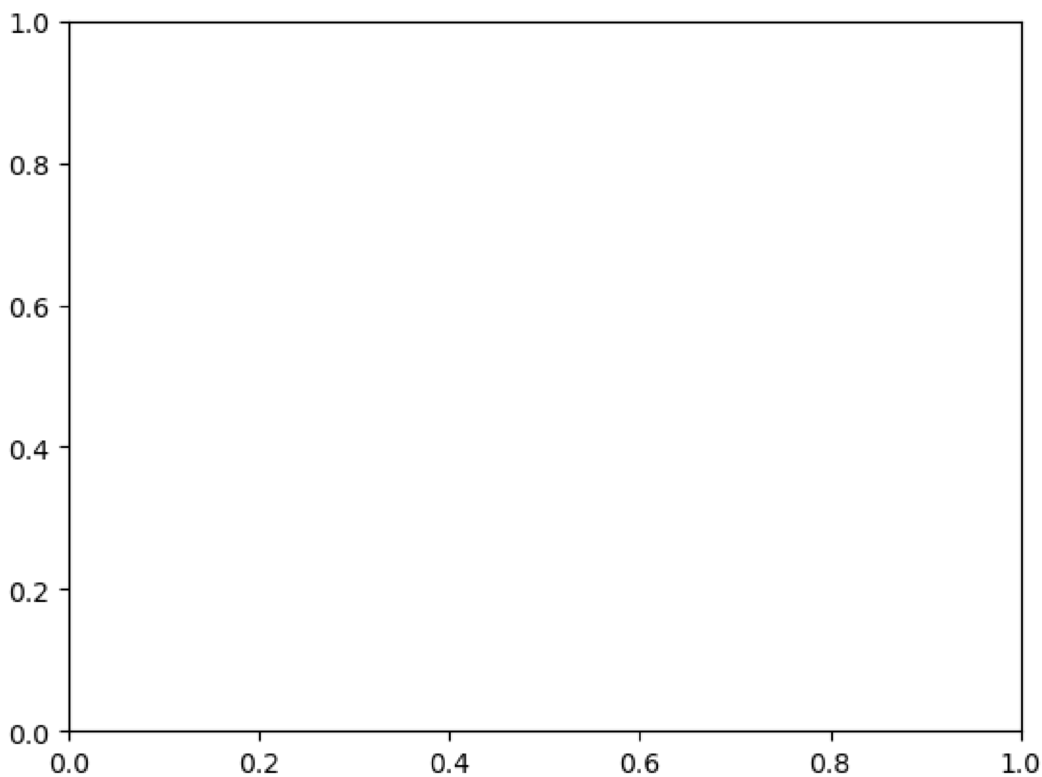
[35]:
```
#6. Calculate and plot the Confidence Intervals for 25 Trials with n=500 and␣
↪CI=95% for ‘Unit price’ column. Observe the results.
print('URK21CS1128')
sample_size = 500
intervals = []
sample_means = []

Cl = 0.95
alpha = (1-CL)/2
z_critical = round(stats.norm.ppf(1-alpha),2)

pm=df['Unit price'].mean()

#25 trials
for sample in range(25):
    sample = np.random.choice(a = df['Unit price'], size = sample_size)
    sample_mean = sample.mean()
    sample_means.append(sample_mean)

    sam_stdev = sample.std()
    margin_of_error = z_critical * (sam_stdev/math.sqrt(sample_size))
    confidence_interval = (sample_mean - margin_of_error,
                           sample_mean + margin_of_error)
    intervals.append(confidence_interval)
print(sample_means)
print(pm)
print(intervals)

plt.errorbar(x = np.arange(0.1, 25, 1),
             y = sample_means,
             yerr = [(top-bot)/2 for top,bot in intervals],
             fmt = 'o')

plt.hlines(xmin=0, xmax=25,
           y=pm,
           linewidth=2.0,
           color="red")
```

```
URK21CS1128
[56.262339999999995, 53.840920000000004, 55.37760000000001, 56.20538,
56.338860000000004, 55.26624, 55.78316, 57.87631999999999, 55.531180000000006,
56.09314, 55.38618, 54.94725999999999, 54.79228, 56.21382, 55.398300000000006,
56.891119999999994, 55.113839999999996, 53.622260000000004, 56.902440000000006,
56.5573, 54.9411, 53.122699999999995, 54.938840000000006, 54.75488000000001,
55.110699999999994]
55.67213
```

```
[(54.34959141456559, 58.1750885854344), (51.98789617172606, 55.69394382827395),
(53.408255464351384, 57.34694453564863), (54.277625356878865,
58.13313464312113), (54.44579980285873, 58.23192019714128), (53.36075274861317,
57.171727251386834), (53.840521208150435, 57.72579879184957),
(55.91333621616036, 59.839303783839625), (53.61959061399828,
57.442769386001736), (54.17653500254634, 58.00974499745366), (53.50070610493708,
57.271653895062926), (53.07762113601136, 56.81689886398863), (52.86297994712205,
56.72158005287795), (54.20915366202474, 58.21848633797526), (53.493806588321355,
57.30279341167866), (54.942762936979804, 58.83947706302018),
(53.133054827704086, 57.094625172295906), (51.70899530226765,
55.535524697732356), (54.98836946081282, 58.81651053918719),
(54.663622911404616, 58.45097708859538), (53.014528416474825,
56.86767158352517), (51.18324781631504, 55.06215218368495), (53.038358586861584,
56.83932141313843), (52.78397104250662, 56.72578895749339), (53.20330456133676,
57.01809543866323)]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[35], line 28
     25 print(pm)
     26 print(intervals)
---> 28 plt.errorbar(x = np.arange(0.1, 25, 1),
     29              y = sample_means,
     30              yerr = [(top-bot)/2 for top,bot in intervals],
     31              fmt = 'o')
     33 plt.hlines(xmin=0, xmax=25,
     34            y=pm,
     35            linewidth=2.0,
     36            color="red")

File /opt/anaconda3/lib/python3.9/site-packages/matplotlib/pyplot.py:2564, in␣
 ↪errorbar(x, y, yerr, xerr, fmt, ecolor, elinewidth, capsize, barsabove,␣
 ↪lolims, uplims, xlolims, xuplims, errorevery, capthick, data, **kwargs)
   2558 @_copy_docstring_and_deprecators(Axes.errorbar)
   2559 def errorbar(
   2560         x, y, yerr=None, xerr=None, fmt='', ecolor=None,
   2561         elinewidth=None, capsize=None, barsabove=False, lolims=False,
   2562         uplims=False, xlolims=False, xuplims=False, errorevery=1,
   2563         capthick=None, *, data=None, **kwargs):
-> 2564     return gca().errorbar(
   2565         x, y, yerr=yerr, xerr=xerr, fmt=fmt, ecolor=ecolor,
   2566         elinewidth=elinewidth, capsize=capsize, barsabove=barsabove,
   2567         lolims=lolims, uplims=uplims, xlolims=xlolims,
   2568         xuplims=xuplims, errorevery=errorevery, capthick=capthick,
   2569         **({"data": data} if data is not None else {}), **kwargs)

File /opt/anaconda3/lib/python3.9/site-packages/matplotlib/__init__.py:1442, in
 ↪_preprocess_data.<locals>.inner(ax, data, *args, **kwargs)
```

```
   1439 @functools.wraps(func)
   1440 def inner(ax, *args, data=None, **kwargs):
   1441     if data is None:
-> 1442         return func(ax, *map(sanitize_sequence, args), **kwargs)
   1444     bound = new_sig.bind(ax, *args, **kwargs)
   1445     auto_label = (bound.arguments.get(label_namer)
   1446                   or bound.kwargs.get(label_namer))

File /opt/anaconda3/lib/python3.9/site-packages/matplotlib/axes/_axes.py:3642, in Axes.errorbar(self, x, y, yerr, xerr, fmt, ecolor, elinewidth, capsize, barsabove, lolims, uplims, xlolims, xuplims, errorevery, capthick, **kwargs)
   3639 res = np.zeros(err.shape, dtype=bool)  # Default in case of nan
   3640 if np.any(np.less(err, -err, out=res, where=(err == err))):
   3641     # like err<0, but also works for timedelta and nan.
-> 3642     raise ValueError(
   3643         f"'{dep_axis}err' must not contain negative values")
   3644 # This is like
   3645 #     elow, ehigh = np.broadcast_to(...)
   3646 #     return dep - elow * ~lolims, dep + ehigh * ~uplims
   3647 # except that broadcast_to would strip units.
   3648 low, high = dep + np.row_stack([-(1 - lolims), 1 - uplims]) * err

ValueError: 'yerr' must not contain negative values
```



[33]:
```
# Print all column names in the DataFrame
print(df.columns)
```

```
Index(['Height', 'Score', 'Age'], dtype='object')
```

[ ]:
```
#7. Calculate the Correlation Coefficient using Pearson for the given table.
print("URK21CS1128")
from scipy.stats import pearsonr
from scipy.stats import spearmanr
import matplotlib.pyplot as plt
x=[17,15,19,17,21]
y=[150,154,169,172,175]
corr, _ = pearsonr(x,y)
print('Pearsons correlation: %.3f' % corr)
```

```
URK21CS1128
Pearsons correlation: 0.721
```

[ ]:
```
#8. Calculate the Correlation Coefficient using Spearman for the given table
print("URK21CS1128")
x=[17,15,19,17,21]
y=[150,154,169,172,175]
corr, _ = spearmanr(x,y)
print('Spearmans correlation: %.3f' % corr)
```

```
URK21CS1128
Spearmans correlation: 0.667
```

[ ]:
```
#9. Calculate the Covariance Matrix for the given data and analyse it
print("URK21CS1128")
import pandas as pd
df = pd.DataFrame(
    {'Height': [64, 66, 68, 69, 73],
     'Score':[580, 570, 590, 660, 600],
     'Age':[29, 33, 37, 46, 55]}
)

cov_matrix = df.cov()
print(cov_matrix)
```

```
URK21CS1128
        Height   Score     Age
Height   11.50    50.0   34.75
Score    50.00  1250.0  205.00
Age      34.75   205.0  110.00
```

[ ]:

```python
#10. Perform a hypothesis testing with Z-test A herd of 1,500 steer was fed a special high-protein grain for a month, has the standard deviation of weight gain for the entire herd was 7.1 and average weight gain per steer for the month was 5 pounds. By feeding the herd with special high-protein grain, it is claimed that the weight of the herd has increased. In order to test this claim, a random sample of 29 were weighed and had gained an average of 6.7 pounds. Can we support the claim at 5 % level?
print("URK21CS1128")
xbar=110
mu=100
n=50
sd=15
z=abs(((xbar-mu)/(sd/math.sqrt(n))))
if(z>1.96):
    print("Reject H0")
else:
    print("Accept H0")
print(z)
```

```
URK21CS1128
Reject H0
4.714045207910317
```

Result:

We have successful executed the program and the output is displayed in the each cell of the code.

Ex 6 Performance Analysis on Regression Techniques

September 11, 2023

[ ]:
```
Exp no: 6
Date: 04/09/2023
```

[ ]:
```
Bewin Felix R A
URK21CS1128
```

[ ]:
```
Aim:
    To execute the functionalities of Performance Analysis on Regression␣
  ↪Techniques using data science.
```

Description:

`Machine learning has two types Supervised and Unsupervised learning. Supervised has Classifica`

Regression: It predicts the continuous output variables based on the independent input variable. like the prediction of house prices based on different parameters like house age, distance from the main road, location, area, etc.

Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between a dependent variable and one or more independent features. When the number of the independent feature, is 1 then it is known as Univariate Linear regression, and in the case of more than one feature, it is known as multivariate linear regression.

Linear regression is one of the most basic types of regression in machine learning. The linear regression model consists of a predictor variable and a dependent variable related linearly to each other. In case the data involves more than one independent variable, then linear regression is called multiple linear regression models.

Simple Linear Regression is a type of Regression algorithms that models the relationship between a dependent variable and a single independent variable. The relationship shown by a Simple Linear Regression model is linear or a sloped straight line, hence it is called Simple Linear Regression.

The key point in Simple Linear Regression is that the dependent variable must be a continuous/real value. However, the independent variable can be measured on continuous or categorical values.

Simple Linear regression algorithm has mainly two objectives:

Model the relationship between the two variables. Such as the relationship between Income and expenditure, experience and Salary, etc. Forecasting new observations. Such as Weather forecasting according to temperature, Revenue of a company according to the investments in a year, etc.

[1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from sklearn.linear_model import LinearRegression
import math
from sklearn.model_selection import train_test_split
```

[2]:
```
#1 a. Calculate the intercept and regression coefficients in y=b0+xb1
print('URK21CS1128')
sub=[1,2,3,4,5,6]
x=[43,21,25,42,57,59]
y=[99,65,79,75,87,81]
x=np.array(x)
y=np.array(y)
m_x=np.mean(x)
m_y=np.mean(y)
xx=x-m_x
yy=y-m_y
xy=xx*yy
xx2=xx*xx

b1=sum(xy)/sum(xx2)
b0=m_y-(b1*m_x)
print('Slope: ',b1)
print('Intercept: ',b0)
y_pred=b0+b1*x
```

```
URK21CS1128
Slope:  0.3852249832102082
Intercept:  65.1415715245131
```

[3]:
```
# b. Analyse the various performance metrics (Mean Squared Error, Mean Absolute␣
↪Error,Root Mean Squared Error, and R-Squared)
print('URK21CS1128')
print('MAE:',mean_absolute_error(y,y_pred))
print('MSE:',mean_squared_error(y,y_pred))
print('RMSE:',math.sqrt(mean_absolute_error(y,y_pred)))
print('R2 score:',r2_score(y,y_pred))
```

```
URK21CS1128
MAE: 7.173852697559885
MSE: 78.64374300425344
RMSE: 2.678404879319011
R2 score: 0.2806974725220722
```

[5]:

```
#2 Develop the linear regression model for the prediction of Graduate␣
↪Admissions from an Indian perspective in-terms of GRE Scores, LOR, CGPA␣
↪using the scikit-learn
df=pd.read_csv('Admission.csv',index_col=0)
x1=df['GRE Score']
x2=df['LOR ']
x3=df['CGPA']
y=df['Chance of Admit ']
# a. Divide the data into training (75%) and testing set (25%)
x1_train,x1_test,y_train,y_test=train_test_split(x1,y,test_size=0.
↪25,random_state=1)
x2_train,x2_test,y_train,y_test=train_test_split(x2,y,test_size=0.
↪25,random_state=1)
x3_train,x3_test,y_train,y_test=train_test_split(x3,y,test_size=0.
↪25,random_state=1)
```

[6]:
```
# b. Display the intercept and regression coefficients for the following cases
#       1. Analyse the impact of GRE scores to the Chance of Admit
#       2. Analyse the impact of LOR to the Chance of Admit
#       3. Analyse the impact of CGPA to the Chance of Admit
print('URK21CS1128')
x1_train=np.array(x1_train).reshape(-1,1)
y_train=np.array(y_train).reshape(-1,1)
model1=LinearRegression()
model1.fit(x1_train,y_train)
print('Impact of GRE Score on Chance of Admit')
print('Regression coefficient= ',model1.coef_)
print('Intercept= ',model1.intercept_,'\n')
x2_train=np.array(x2_train).reshape(-1,1)
y_train=np.array(y_train).reshape(-1,1)
model2=LinearRegression()
model2.fit(x2_train,y_train)
print('Impact of LOR on Chance of Admit')
print('Regression coefficient= ',model2.coef_)
print('Intercept= ',model2.intercept_,'\n')
x3_train=np.array(x3_train).reshape(-1,1)
y_train=np.array(y_train).reshape(-1,1)
model3=LinearRegression()
model3.fit(x3_train,y_train)
print('Impact of CGPA on Chance of Admit')
print('Regression coefficient= ',model3.coef_)
print('Intercept= ',model3.intercept_)
```

```
URK21CS1128
Impact of GRE Score on Chance of Admit
Regression coefficient=  [[0.01004167]]
Intercept=  [-2.45230421]
```

```
Impact of LOR on Chance of Admit
Regression coefficient=  [[0.09357372]]
Intercept=  [0.39662437]

Impact of CGPA on Chance of Admit
Regression coefficient=  [[0.20599061]]
Intercept=  [-1.04359588]
```

[7]:
```python
# c. Predict the y value (y') for the testing set (x) and analyse the
↪performance metrics with the actual value (y) and predicted values (y') for
↪the above 3 cases
x1_test=np.array(x1_test).reshape(-1,1)
x2_test=np.array(x2_test).reshape(-1,1)
x3_test=np.array(x3_test).reshape(-1,1)
y1_p=model1.predict(x1_test)
y2_p=model2.predict(x2_test)
y3_p=model3.predict(x3_test)
y_test=np.array(y_test).reshape(-1,1)
```

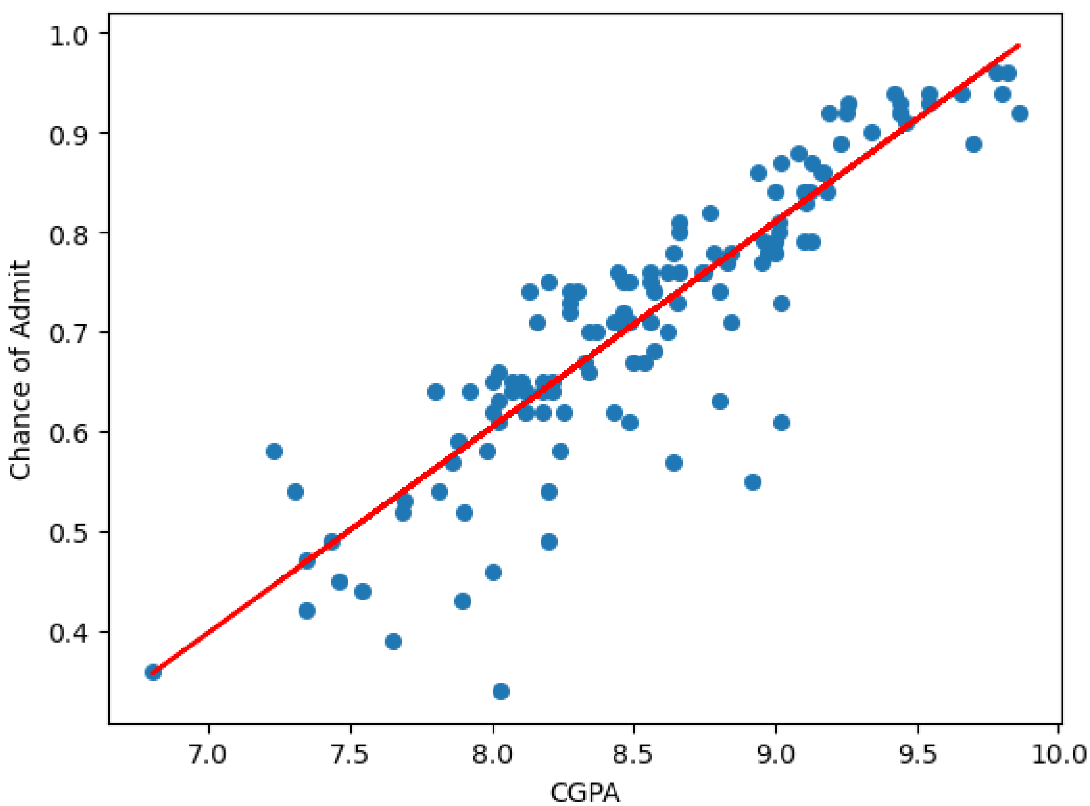
[8]:
```python
print('URK21CS1128\n')
print('Impact of GRE Score on Chance of Admit')
print('MAE:',mean_absolute_error(y_test,y1_p))
print('MSE:',mean_squared_error(y_test,y1_p))
print('RMSE:',math.sqrt(mean_absolute_error(y_test,y1_p)))
print('R2 score:',r2_score(y_test,y1_p),'\n')
print('Impact of LOR on Chance of Admit')
print('MAE:',mean_absolute_error(y_test,y2_p))
print('MSE:',mean_squared_error(y_test,y2_p))
print('RMSE:',math.sqrt(mean_absolute_error(y_test,y2_p)))
print('R2 score:',r2_score(y_test,y2_p),'\n')
print('Impact of CGPA on Chance of Admit')
print('MAE:',mean_absolute_error(y_test,y3_p))
print('MSE:',mean_squared_error(y_test,y3_p))
print('RMSE:',math.sqrt(mean_absolute_error(y_test,y3_p)))
print('R2 score:',r2_score(y_test,y3_p))
```

```
URK21CS1128

Impact of GRE Score on Chance of Admit
MAE: 0.06264911695468035
MSE: 0.007625959038845016
RMSE: 0.25029805623432305
R2 score: 0.6260055317126936

Impact of LOR on Chance of Admit
MAE: 0.08129920544256705
MSE: 0.010893335796659088
```

```
RMSE: 0.2851301552669711
R2 score: 0.4657659045381354

Impact of CGPA on Chance of Admit
MAE: 0.04514566173687265
MSE: 0.004336070707235971
RMSE: 0.21247508497909268
R2 score: 0.7873491779396589
```

[9]:
```python
# d. Identify the input parameter that has a greater impact in the prediction␣
 ↪of Graduate Admissions from an Indian perspective
print('URK21CS1128')
print('CGPA has the highest impact on the prediction of graduate admissions')
# e. Plot the regression line for the input parameter that has a greater impact␣
 ↪in the prediction of prediction of Graduate Admissions from an Indian␣
 ↪perspective
plt.scatter(x3_test,y_test,marker='o',s=30)
plt.plot(x3_test,y3_p,c='red')
plt.xlabel('CGPA')
plt.ylabel('Chance of Admit')
plt.show()
```

```
URK21CS1128
CGPA has the highest impact on the prediction of graduate admissions
```



[ ]:

```
Result:
    The functionalities of Performance Analysis on Regression Techniques using␣
 ↪data science were executed successfully
```

Ex-7 Perfomance analysis on KNN classification technique

October 26, 2023

URK21CS1128

AIM: To demonstrate performance analysis on KNN classification technique

DESCRIPTION: K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. The following two properties would define KNN well

K-nearest neighbors (KNN) algorithm uses feature similarity to predict the values of new datapoints which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of following steps:

Step1: For implementing any algorithm, we need dataset. So, during the first step of KNN, load the training as well as test data. Step2: Choose the value of K i.e. the nearest data points. K can be any integer. Step3: For each point in the test data do the following:

3.1: Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean. 3.2: Now, based on the distance value, sort them in ascending order. 3.3: Next, it will choose the top K rows from the sorted array. 3.4: Now, it will assign a class to the test point based on most frequent class of these rows. Step4: End Specificity, in contrast to recall, may be defined as the number of negatives returned by the classification model.

Support may be defined as the number of samples of the true response that lies in each class of target values. F1 Score This score will give us the harmonic mean of precision and recall.

Mathematically, F1 score is the weighted average of the precision and recall. The best value of F1 would be 1 and worst would be 0. F1 score will be calculated with the help of following formula: F1 = 2 ∗ (precision ∗ recall) / (precision + recall) F1 score is having equal relative contribution of precision and recall. classification_report function of sklearn.metrics is used to get the classification report of classification model. AUC (Area Under Curve)-ROC (Receiver Operating Characteristic) is a performance metric, based on varying threshold values, for classification problems. As name suggests, ROC is a prob- ability curve and AUC measure the separability. In simple words, AUC-ROC metric will tell us about the capability of model in distinguishing the classes. Higher the AUC, better the model. Mathematically, it can be created by plotting TPR (True Positive Rate) i.e. Sensitivity or recall vs FPR (False Positive Rate) i.e. 1-Specificity, at various threshold values. roc_auc_score function of sklearn.metrics is used to compute AUC-ROC.

[3]:
```
import pandas as pd
import numpy as np
```

```python
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix,␣
↪accuracy_score,classification_report, roc_auc_score
```

1. Develop a KNN classification model for the wine dataset using the scikit-learn

a. Read the data

[4]:
```python
print("URK21CS1128")
data = pd.read_csv('wine.csv')
data.head(2)
```

```
URK21CS1128
```

[4]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70          0.0             1.9      0.076
1            7.8              0.88          0.0             2.6      0.098

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                    34   0.9978  3.51       0.56
1                 25.0                    67   0.9968  3.20       0.68

   alcohol quality
0      9.4     bad
1      9.8     bad
```

b. Data Cleaning
    a. Replace 0 in ['chlorides', 'density', 'pH', 'sulphates'] column with NaNvalue
c. Identify the columns with null value
d. Filling the null values by imputing the mean values in the corresponding column

[7]:
```python
cols_to_replace_zero = ['chlorides', 'density', 'pH', 'sulphates']
data[cols_to_replace_zero] = data[cols_to_replace_zero].replace(0, float('nan'))
columns_with_null = data.columns[data.isnull().any()].tolist()
data[columns_with_null] = data[columns_with_null].fillna(data.
↪mean(numeric_only=True))
data.head(10)
```

[7]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076
5            7.4              0.66         0.00             1.8      0.075
6            7.9              0.60         0.06             1.6      0.069
7            7.3              0.65         0.00             1.2      0.065
8            7.8              0.58         0.02             2.0      0.073
```

```
9             7.5              0.50         0.36             6.1      0.071

   free sulfur dioxide  total sulfur dioxide  density        pH  sulphates  \
0                 11.0                    34   0.9978  3.510000       0.56
1                 25.0                    67   0.9968  3.200000       0.68
2                 15.0                    54   0.9970  3.260000       0.65
3                 17.0                    60   0.9980  3.160000       0.58
4                 11.0                    34   0.9978  3.510000       0.56
5                 13.0                    40   0.9978  3.510000       0.56
6                 15.0                    59   0.9964  3.299488       0.46
7                 15.0                    21   0.9946  3.390000       0.47
8                  9.0                    18   0.9968  3.360000       0.57
9                 17.0                   102   0.9978  3.350000       0.80

   alcohol quality
0      9.4     bad
1      9.8     bad
2      9.8     bad
3      9.8    good
4      9.4     bad
5      9.4     bad
6      9.4     bad
7     10.0    good
8      9.5    good
9     10.5     bad
```

c. Use the columns: ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide'] as the independent variables

[8]:
```
print("URK21CS1128")
X = data[['fixed acidity', 'volatile acidity', 'citric acid', 'residual␣
↪sugar','chlorides', 'free sulfur dioxide']]
X.head(2)
```

```
URK21CS1128
```

[8]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70          0.0             1.9      0.076
1            7.8              0.88          0.0             2.6      0.098

   free sulfur dioxide
0                 11.0
1                 25.0
```

d. Use the target variable as 'quality' ('good' and 'bad' based on score >5 and <5)

[9]:
```
print("URK21CS1128")
y = data['quality']
y.head(4)
```

```
URK21CS1128
```

[9]:
```
0     bad
1     bad
2     bad
3    good
Name: quality, dtype: object
```

e. Encode the categorical value of the target column to numerical value

[10]:
```python
print("URK21CS1128")
y = y.replace({'good':0, 'bad':1})
y.head(4)
```

```
URK21CS1128
```

[10]:
```
0    1
1    1
2    1
3    0
Name: quality, dtype: int64
```

f. Divide the data into training (75%) and testing set (25%)

[11]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪25,random_state=42)
```

g. Perform the classification with K=3

[12]:
```python
print("URK21CS1128")
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

```
URK21CS1128
```

[12]:
```
KNeighborsClassifier(n_neighbors=3)
```

h. Analyse the performance of the classifier with various performance measures␣and display such as confusion matrix, accuracy, recall, precision,␣specificity, f-score, Receiver operating characteristic (ROC) curve and Area␣Under Curve (AUC) score

[13]:
```python
print("URK21CS1128")
y_pred = knn.predict(X_test)
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("AUC Score:", roc_auc_score(y_test, y_pred))
```

```
URK21CS1128
Confusion Matrix:
 [[73 61]
 [37 79]]
```

```
Accuracy: 0.608
Classification Report:
               precision    recall  f1-score   support

           0       0.66      0.54      0.60       134
           1       0.56      0.68      0.62       116

    accuracy                           0.61       250
   macro avg       0.61      0.61      0.61       250
weighted avg       0.62      0.61      0.61       250

AUC Score: 0.6129053010808028
```

i. Change the value of K in KNN with 5,7,9,11 and tabulate the various TP, TN,␣ accuracy, f-score and AUC score obtained

[14]:
```python
print("URK21CS1128")
for i in [5,7,9,11]:
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))
    print("AUC Score:", roc_auc_score(y_test, y_pred))
```

```
URK21CS1128
Confusion Matrix:
 [[68 66]
 [46 70]]
Accuracy: 0.552
Classification Report:
               precision    recall  f1-score   support

           0       0.60      0.51      0.55       134
           1       0.51      0.60      0.56       116

    accuracy                           0.55       250
   macro avg       0.56      0.56      0.55       250
weighted avg       0.56      0.55      0.55       250

AUC Score: 0.5554554812146166
Confusion Matrix:
 [[64 70]
 [43 73]]
Accuracy: 0.548
Classification Report:
               precision    recall  f1-score   support
```

```
           0       0.60      0.48      0.53       134
           1       0.51      0.63      0.56       116

    accuracy                           0.55       250
   macro avg       0.55      0.55      0.55       250
weighted avg       0.56      0.55      0.55       250

AUC Score: 0.5534611425630469
Confusion Matrix:
 [[68 66]
 [42 74]]
Accuracy: 0.568
Classification Report:
               precision    recall  f1-score   support

           0       0.62      0.51      0.56       134
           1       0.53      0.64      0.58       116

    accuracy                           0.57       250
   macro avg       0.57      0.57      0.57       250
weighted avg       0.58      0.57      0.57       250

AUC Score: 0.5726968605249615
Confusion Matrix:
 [[73 61]
 [47 69]]
Accuracy: 0.568
Classification Report:
               precision    recall  f1-score   support

           0       0.61      0.54      0.57       134
           1       0.53      0.59      0.56       116

    accuracy                           0.57       250
   macro avg       0.57      0.57      0.57       250
weighted avg       0.57      0.57      0.57       250

AUC Score: 0.5698018528049409
```

0.1 j. Analyse and infer for which K value, the classification algorithm provides␣

better performance.

K = 5 provides the highest accuracy and highest AUC score. K = 7 provides the second-highest accuracy and the second-highest AUC score. K = 9 provides the third-highest accuracy and the third-highest AUC score. K = 11 provides the lowest accuracy and but the lowest AUC score. K = 5 provides performance for this classification task. Therefore, K = 5 is considered the preferred choice for this specific problem.Therefore, we can infer that K = 5 provides better performance for this particular dataset ,classification problem.

[ ]:
```
Result : Thus the program has been executed and the output has been verified␣
↪successfully
```

Ex.8 Performance Analysis on Decision Tree Classification Technique

October 25, 2023

[ ]:
```
Bewin Felix R A
URK21CS1128
```

[ ]:
```
AIM:
    To analyse the performance of Decision Tree Classification Technique.
```

[ ]:
```
DESCRIPTION:
    Decision Tree is a Supervised learning technique that can be used for both
classification and Regression problems, but mostly it is preferred for solving
Classification problems. It is a tree-structured classifier, where internal
nodes represent the features of a dataset, branches represent the decision
rules and each leaf node represents the outcome.
In a Decision tree, there are two nodes, which are the Decision Node and Leaf
Node. Decision nodes are used to make any decision and have multiple branches,
whereas Leaf nodes are the output of those decisions and do not contain any
further branches.The decisions or the test are performed on the basis of
features of the given dataset. It is a graphical representation for getting
all the possible solutions to a problem/decision based on given conditions.
It is called a decision tree because, similar to a tree, it starts with the
root node, which expands on further branches and constructs a tree-like
structure. In order to build a tree, we use the CART algorithm, which stands
for Classification and Regression Tree algorithm. A decision tree simply asks
a question, and based on the answer (Yes/No), it further split the tree into
subtrees.
Decision Tree Terminologies
Root Node: Root node is from where the decision tree starts. It represents the
entire dataset, which further gets divided into two or more homogeneous sets.
Leaf Node: Leaf nodes are the final output node, and the tree cannot be
segregated further after getting a leaf node.
Splitting: Splitting is the process of dividing the decision node/root nodE
into sub-nodes according to the given conditions.
Branch/Sub Tree: A tree formed by splitting the tree.
Pruning: Pruning is the process of removing the unwanted branches from the tree.
Parent/Child node: The root node of the tree is called the parent node, and
other nodes are called the child nodes.
Algorithm:
```

```
Step-1: Begin the tree with the root node, says S, which contains the complete
dataset.
Step-2: Find the best attribute in the dataset using Attribute Selection
Measure (ASM).
Step-3: Divide the S into subsets that contains possible values for the best
attributes.
Step-4: Generate the decision tree node, which contains the best attribute.
Step-5: Recursively make new decision trees using the subsets of the dataset
created in step3.
Continue this process until a stage is reached where you cannot further
classify the nodes and called the final node as a leaf node.
```

[ ]:
```
1. Develop a Decision Tree classification model for the Social_Network dataset
 ↪using the
scikit-learn
```

[3]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import confusion_matrix, accuracy_score,
 ↪precision_score,f1_score, recall_score
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.metrics import roc_auc_score
```

[4]:
```
print('1128')
df = pd.read_csv('Social_Network.csv')
df
```

1128

[4]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15668575	0	26	43000	No
1	15603246	0	27	57000	No
2	15598044	0	27	84000	No
3	15727311	0	35	65000	No
4	15570769	0	26	80000	No
..	…	…	…	…	…
395	15672330	1	47	34000	Yes
396	15807837	1	48	33000	Yes
397	15592570	1	47	23000	Yes
398	15635893	1	60	42000	Yes
399	15706071	1	51	23000	Yes

[400 rows x 5 columns]

[5]:
```python
#a. Use the columns: 'Gender', 'Age', 'EstimatedSalary' as the independent␣
↪variables
print(1128)
x=df[['Gender', 'Age', 'EstimatedSalary']]
x
```

```
1128
```

[5]:
```
     Gender  Age  EstimatedSalary
0         0   26            43000
1         0   27            57000
2         0   27            84000
3         0   35            65000
4         0   26            80000
..      …    …              …
395       1   47            34000
396       1   48            33000
397       1   47            23000
398       1   60            42000
399       1   51            23000

[400 rows x 3 columns]
```

[6]:
```python
# b. Use the target variable as 'Purchased' (Yes-Y, No-N).
print(1128)
y = df['Purchased']
y
```

```
1128
```

[6]:
```
0       No
1       No
2       No
3       No
4       No
      …
395    Yes
396    Yes
397    Yes
398    Yes
399    Yes
Name: Purchased, Length: 400, dtype: object
```

[7]:
```python
# c. Encode the categorical value of the target column to numerical value.
print(1128)
y.replace('Yes',1 ,inplace=True)
y.replace('No',0,inplace=True)
y
```

1128

[7]:
```
0      0
1      0
2      0
3      0
4      0
      ..
395    1
396    1
397    1
398    1
399    1
Name: Purchased, Length: 400, dtype: int64
```

[8]:
```
#d. Divide the data into training (75%) and testing set (25%).
print(1128)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
 ↪25,random_state=1)
```

1128

[9]:
```
# e. Perform the classification with entropy and information gain as decision␣
 ↪criteria in
#decision tree
print(1128)
tree_entropy = DecisionTreeClassifier(criterion='entropy')
tree_entropy.fit(x_train,y_train)
y_pred = tree_entropy.predict(x_test)
tree_ig = DecisionTreeClassifier(criterion='gini')
tree_ig.fit(x_train,y_train)
y_pred_ig = tree_ig.predict(x_test)
```

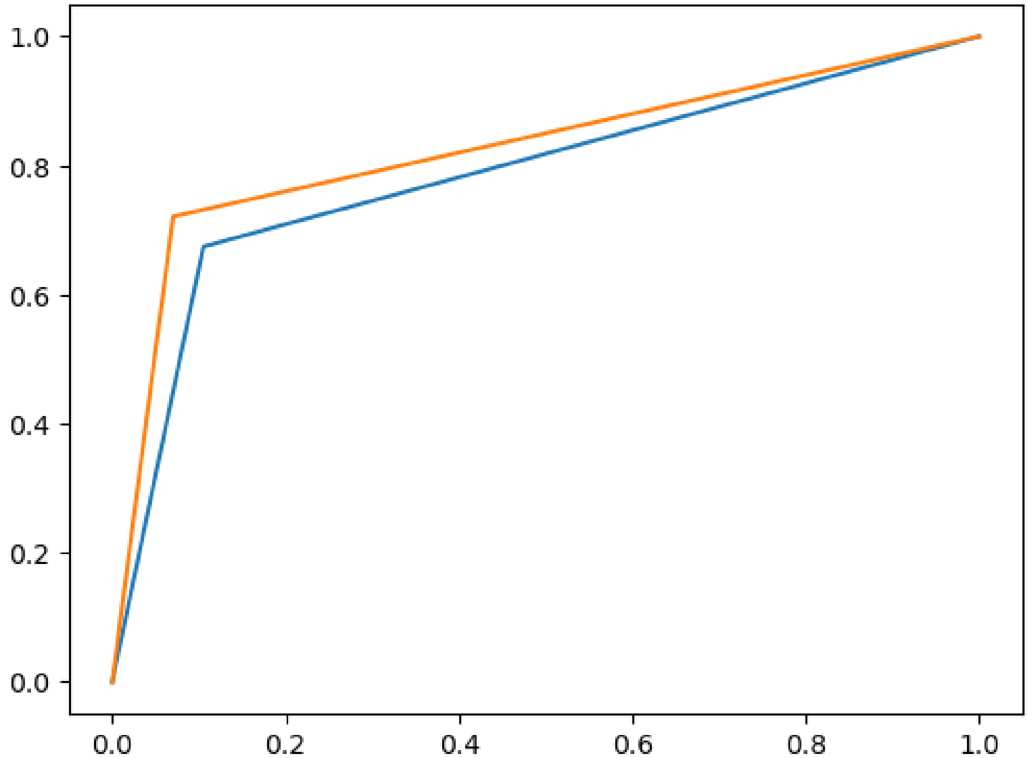
1128

[10]:
```
# f. Analyse the performance of the classifier with various performance␣
 ↪measures such as
#confusion matrix, accuracy, recall, precision, specificity, f-score, Receiver␣
 ↪operating
#characteristic (ROC) curve and Area Under Curve (AUC) score
print(1128)
print('Accuracy using Entropy : ',accuracy_score(y_test,y_pred))
print('Accuracy using ig : ',accuracy_score(y_test,y_pred_ig))

print('Recall using Entropy : ',recall_score(y_test,y_pred))
print('Recall using ig : ',recall_score(y_test,y_pred_ig))

print('Precision using Entropy : ',precision_score(y_test,y_pred))
```

```python
print('Precision using ig : ',precision_score(y_test,y_pred_ig))

print('Specificity using Entropy : ',recall_score(y_test,y_pred,pos_label=0))
print('Specificity using ig : ',recall_score(y_test,y_pred_ig,pos_label=0))

print('F1_score using Entropy : ',f1_score(y_test,y_pred))
print('F1_score using ig : ',f1_score(y_test,y_pred_ig))

print('AUC score using Entropy : ',roc_auc_score(y_test,y_pred))
print('AUC score using ig : ',roc_auc_score(y_test,y_pred_ig))

# ROC curve entropy
fpr, tpr, _ = roc_curve(y_test, y_pred)
plt.plot(fpr,tpr)

#ROC curve ig
fpr, tpr, _ = roc_curve(y_test, y_pred_ig)
plt.plot(fpr,tpr)
```

```
1128
Accuracy using Entropy :  0.8
Accuracy using ig :  0.84
Recall using Entropy :  0.6744186046511628
Recall using ig :  0.7209302325581395
Precision using Entropy :  0.8285714285714286
Precision using ig :  0.8857142857142857
Specificity using Entropy :  0.8947368421052632
Specificity using ig :  0.9298245614035088
F1_score using Entropy :  0.7435897435897435
F1_score using ig :  0.7948717948717948
AUC score using Entropy :  0.7845777233782129
AUC score using ig :  0.8253773969808241
```

```
[10]: [<matplotlib.lines.Line2D at 0x7f56e214f730>]
```



[11]:
```python
# g. Display the constructed decision tree sklearn.tree.plot_tree method.
print(1128)
plot_tree(tree_entropy)
```

```
1128
```

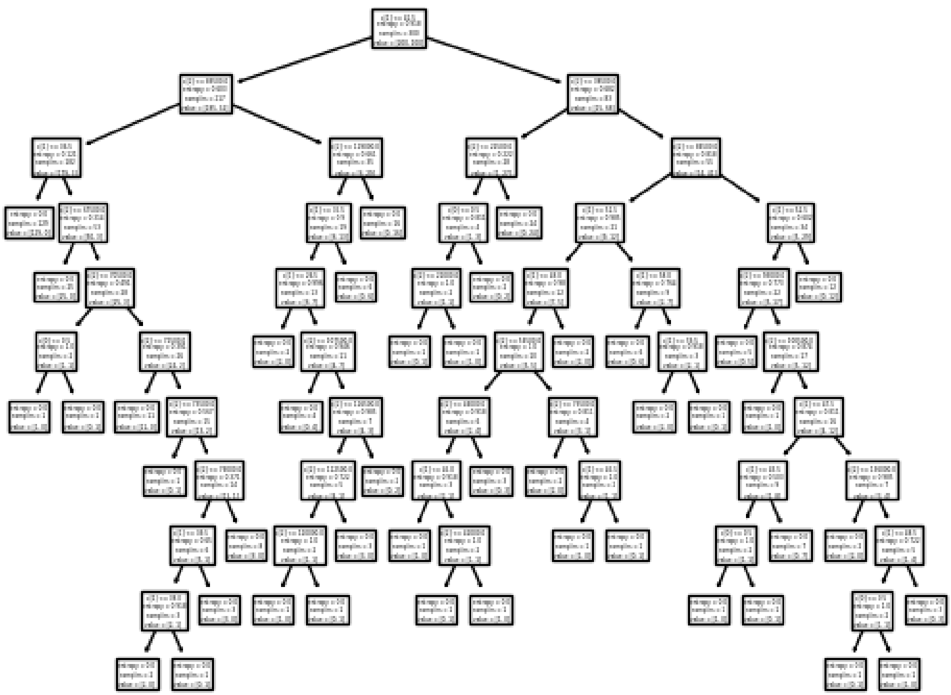
[11]:
```
[Text(0.41785714285714287, 0.9545454545454546, 'x[1] <= 42.5\nentropy =
0.918\nsamples = 300\nvalue = [200, 100]'),
 Text(0.21428571428571427, 0.8636363636363636, 'x[2] <= 89500.0\nentropy =
0.603\nsamples = 217\nvalue = [185, 32]'),
 Text(0.05714285714285714, 0.7727272727272727, 'x[1] <= 36.5\nentropy =
0.121\nsamples = 182\nvalue = [179, 3]'),
 Text(0.02857142857142857, 0.6818181818181818, 'entropy = 0.0\nsamples =
129\nvalue = [129, 0]'),
 Text(0.08571428571428572, 0.6818181818181818, 'x[2] <= 67500.0\nentropy =
0.314\nsamples = 53\nvalue = [50, 3]'),
 Text(0.05714285714285714, 0.5909090909090909, 'entropy = 0.0\nsamples =
25\nvalue = [25, 0]'),
 Text(0.11428571428571428, 0.5909090909090909, 'x[2] <= 70500.0\nentropy =
0.491\nsamples = 28\nvalue = [25, 3]'),
 Text(0.05714285714285714, 0.5, 'x[0] <= 0.5\nentropy = 1.0\nsamples = 2\nvalue
= [1, 1]'),
```

```
 Text(0.02857142857142857, 0.4090909090909091, 'entropy = 0.0\nsamples =
1\nvalue = [1, 0]'),
 Text(0.08571428571428572, 0.4090909090909091, 'entropy = 0.0\nsamples =
1\nvalue = [0, 1]'),
 Text(0.17142857142857143, 0.5, 'x[2] <= 72500.0\nentropy = 0.391\nsamples =
26\nvalue = [24, 2]'),
 Text(0.14285714285714285, 0.4090909090909091, 'entropy = 0.0\nsamples =
11\nvalue = [11, 0]'),
 Text(0.2, 0.4090909090909091, 'x[2] <= 73500.0\nentropy = 0.567\nsamples =
15\nvalue = [13, 2]'),
 Text(0.17142857142857143, 0.3181818181818182, 'entropy = 0.0\nsamples =
1\nvalue = [0, 1]'),
 Text(0.22857142857142856, 0.3181818181818182, 'x[2] <= 76000.0\nentropy =
0.371\nsamples = 14\nvalue = [13, 1]'),
 Text(0.2, 0.22727272727272727, 'x[1] <= 39.5\nentropy = 0.65\nsamples =
6\nvalue = [5, 1]'),
 Text(0.17142857142857143, 0.13636363636363635, 'x[1] <= 38.0\nentropy =
0.918\nsamples = 3\nvalue = [2, 1]'),
 Text(0.14285714285714285, 0.045454545454545456, 'entropy = 0.0\nsamples =
2\nvalue = [2, 0]'),
 Text(0.2, 0.045454545454545456, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.22857142857142856, 0.13636363636363635, 'entropy = 0.0\nsamples =
3\nvalue = [3, 0]'),
 Text(0.2571428571428571, 0.22727272727272727, 'entropy = 0.0\nsamples =
8\nvalue = [8, 0]'),
 Text(0.37142857142857144, 0.7727272727272727, 'x[2] <= 119000.0\nentropy =
0.661\nsamples = 35\nvalue = [6, 29]'),
 Text(0.34285714285714286, 0.6818181818181818, 'x[1] <= 35.5\nentropy =
0.9\nsamples = 19\nvalue = [6, 13]'),
 Text(0.3142857142857143, 0.5909090909090909, 'x[1] <= 26.5\nentropy =
0.996\nsamples = 13\nvalue = [6, 7]'),
 Text(0.2857142857142857, 0.5, 'entropy = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.34285714285714286, 0.5, 'x[2] <= 107500.0\nentropy = 0.946\nsamples =
11\nvalue = [4, 7]'),
 Text(0.3142857142857143, 0.4090909090909091, 'entropy = 0.0\nsamples = 4\nvalue
= [0, 4]'),
 Text(0.37142857142857144, 0.4090909090909091, 'x[2] <= 116500.0\nentropy =
0.985\nsamples = 7\nvalue = [4, 3]'),
 Text(0.34285714285714286, 0.3181818181818182, 'x[2] <= 112500.0\nentropy =
0.722\nsamples = 5\nvalue = [4, 1]'),
 Text(0.3142857142857143, 0.22727272727272727, 'x[2] <= 110000.0\nentropy =
1.0\nsamples = 2\nvalue = [1, 1]'),
 Text(0.2857142857142857, 0.13636363636363635, 'entropy = 0.0\nsamples =
1\nvalue = [1, 0]'),
 Text(0.34285714285714286, 0.13636363636363635, 'entropy = 0.0\nsamples =
1\nvalue = [0, 1]'),
 Text(0.37142857142857144, 0.22727272727272727, 'entropy = 0.0\nsamples =
```

```
3\nvalue = [3, 0]'),
 Text(0.4, 0.3181818181818182, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.37142857142857144, 0.5909090909090909, 'entropy = 0.0\nsamples =
6\nvalue = [0, 6]'),
 Text(0.4, 0.6818181818181818, 'entropy = 0.0\nsamples = 16\nvalue = [0, 16]'),
 Text(0.6214285714285714, 0.8636363636363636, 'x[2] <= 38500.0\nentropy =
0.682\nsamples = 83\nvalue = [15, 68]'),
 Text(0.5142857142857142, 0.7727272727272727, 'x[2] <= 22500.0\nentropy =
0.222\nsamples = 28\nvalue = [1, 27]'),
 Text(0.4857142857142857, 0.6818181818181818, 'x[0] <= 0.5\nentropy =
0.811\nsamples = 4\nvalue = [1, 3]'),
 Text(0.45714285714285713, 0.5909090909090909, 'x[2] <= 21000.0\nentropy =
1.0\nsamples = 2\nvalue = [1, 1]'),
 Text(0.42857142857142855, 0.5, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.4857142857142857, 0.5, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.5142857142857142, 0.5909090909090909, 'entropy = 0.0\nsamples = 2\nvalue
= [0, 2]'),
 Text(0.5428571428571428, 0.6818181818181818, 'entropy = 0.0\nsamples =
24\nvalue = [0, 24]'),
 Text(0.7285714285714285, 0.7727272727272727, 'x[2] <= 84500.0\nentropy =
0.818\nsamples = 55\nvalue = [14, 41]'),
 Text(0.6285714285714286, 0.6818181818181818, 'x[1] <= 51.5\nentropy =
0.985\nsamples = 21\nvalue = [9, 12]'),
 Text(0.5714285714285714, 0.5909090909090909, 'x[1] <= 48.0\nentropy =
0.98\nsamples = 12\nvalue = [7, 5]'),
 Text(0.5428571428571428, 0.5, 'x[2] <= 54500.0\nentropy = 1.0\nsamples =
10\nvalue = [5, 5]'),
 Text(0.4857142857142857, 0.4090909090909091, 'x[2] <= 44000.0\nentropy =
0.918\nsamples = 6\nvalue = [2, 4]'),
 Text(0.45714285714285713, 0.3181818181818182, 'x[1] <= 45.0\nentropy =
0.918\nsamples = 3\nvalue = [2, 1]'),
 Text(0.42857142857142855, 0.22727272727272727, 'entropy = 0.0\nsamples =
1\nvalue = [1, 0]'),
 Text(0.4857142857142857, 0.22727272727272727, 'x[2] <= 42000.0\nentropy =
1.0\nsamples = 2\nvalue = [1, 1]'),
 Text(0.45714285714285713, 0.13636363636363635, 'entropy = 0.0\nsamples =
1\nvalue = [0, 1]'),
 Text(0.5142857142857142, 0.13636363636363635, 'entropy = 0.0\nsamples =
1\nvalue = [1, 0]'),
 Text(0.5142857142857142, 0.3181818181818182, 'entropy = 0.0\nsamples = 3\nvalue
= [0, 3]'),
 Text(0.6, 0.4090909090909091, 'x[2] <= 76500.0\nentropy = 0.811\nsamples =
4\nvalue = [3, 1]'),
 Text(0.5714285714285714, 0.3181818181818182, 'entropy = 0.0\nsamples = 2\nvalue
= [2, 0]'),
 Text(0.6285714285714286, 0.3181818181818182, 'x[1] <= 45.5\nentropy =
1.0\nsamples = 2\nvalue = [1, 1]'),
```

```
 Text(0.6, 0.22727272727272727, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.6571428571428571, 0.22727272727272727, 'entropy = 0.0\nsamples =
1\nvalue = [0, 1]'),
 Text(0.6, 0.5, 'entropy = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.6857142857142857, 0.5909090909090909, 'x[1] <= 58.0\nentropy =
0.764\nsamples = 9\nvalue = [2, 7]'),
 Text(0.6571428571428571, 0.5, 'entropy = 0.0\nsamples = 6\nvalue = [0, 6]'),
 Text(0.7142857142857143, 0.5, 'x[1] <= 59.5\nentropy = 0.918\nsamples =
3\nvalue = [2, 1]'),
 Text(0.6857142857142857, 0.4090909090909091, 'entropy = 0.0\nsamples = 2\nvalue
= [2, 0]'),
 Text(0.7428571428571429, 0.4090909090909091, 'entropy = 0.0\nsamples = 1\nvalue
= [0, 1]'),
 Text(0.8285714285714286, 0.6818181818181818, 'x[1] <= 52.5\nentropy =
0.602\nsamples = 34\nvalue = [5, 29]'),
 Text(0.8, 0.5909090909090909, 'x[2] <= 93000.0\nentropy = 0.773\nsamples =
22\nvalue = [5, 17]'),
 Text(0.7714285714285715, 0.5, 'entropy = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(0.8285714285714286, 0.5, 'x[2] <= 100500.0\nentropy = 0.874\nsamples =
17\nvalue = [5, 12]'),
 Text(0.8, 0.4090909090909091, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.8571428571428571, 0.4090909090909091, 'x[1] <= 47.5\nentropy =
0.811\nsamples = 16\nvalue = [4, 12]'),
 Text(0.8, 0.3181818181818182, 'x[1] <= 43.5\nentropy = 0.503\nsamples =
9\nvalue = [1, 8]'),
 Text(0.7714285714285715, 0.22727272727272727, 'x[0] <= 0.5\nentropy =
1.0\nsamples = 2\nvalue = [1, 1]'),
 Text(0.7428571428571429, 0.13636363636363635, 'entropy = 0.0\nsamples =
1\nvalue = [1, 0]'),
 Text(0.8, 0.13636363636363635, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.8285714285714286, 0.22727272727272727, 'entropy = 0.0\nsamples =
7\nvalue = [0, 7]'),
 Text(0.9142857142857143, 0.3181818181818182, 'x[2] <= 136000.0\nentropy =
0.985\nsamples = 7\nvalue = [3, 4]'),
 Text(0.8857142857142857, 0.22727272727272727, 'entropy = 0.0\nsamples =
2\nvalue = [2, 0]'),
 Text(0.9428571428571428, 0.22727272727272727, 'x[1] <= 48.5\nentropy =
0.722\nsamples = 5\nvalue = [1, 4]'),
 Text(0.9142857142857143, 0.13636363636363635, 'x[0] <= 0.5\nentropy =
1.0\nsamples = 2\nvalue = [1, 1]'),
 Text(0.8857142857142857, 0.045454545454545456, 'entropy = 0.0\nsamples =
1\nvalue = [0, 1]'),
 Text(0.9428571428571428, 0.045454545454545456, 'entropy = 0.0\nsamples =
1\nvalue = [1, 0]'),
 Text(0.9714285714285714, 0.13636363636363635, 'entropy = 0.0\nsamples =
3\nvalue = [0, 3]'),
 Text(0.8571428571428571, 0.5909090909090909, 'entropy = 0.0\nsamples =
```

12\nvalue = [0, 12]')]



[12]:
```
print(1128)
plot_tree(tree_ig)
```

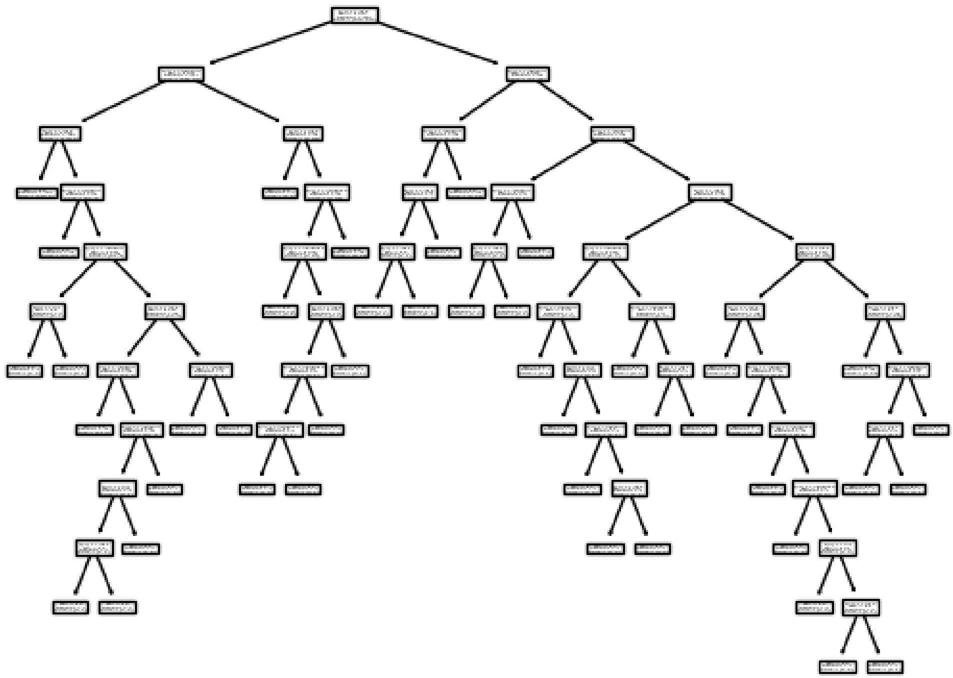
1128

[12]:
```
[Text(0.3704268292682927, 0.9583333333333334, 'x[1] <= 42.5\ngini =
0.444\nsamples = 300\nvalue = [200, 100]'),
 Text(0.18902439024390244, 0.875, 'x[2] <= 89500.0\ngini = 0.251\nsamples =
217\nvalue = [185, 32]'),
 Text(0.06097560975609756, 0.7916666666666666, 'x[1] <= 36.5\ngini =
0.032\nsamples = 182\nvalue = [179, 3]'),
 Text(0.036585365853658534, 0.7083333333333334, 'gini = 0.0\nsamples =
129\nvalue = [129, 0]'),
 Text(0.08536585365853659, 0.7083333333333334, 'x[2] <= 67500.0\ngini =
0.107\nsamples = 53\nvalue = [50, 3]'),
 Text(0.06097560975609756, 0.625, 'gini = 0.0\nsamples = 25\nvalue = [25, 0]'),
 Text(0.10975609756097561, 0.625, 'x[2] <= 70500.0\ngini = 0.191\nsamples =
28\nvalue = [25, 3]'),
 Text(0.04878048780487805, 0.5416666666666666, 'x[0] <= 0.5\ngini = 0.5\nsamples
= 2\nvalue = [1, 1]'),
 Text(0.024390243902439025, 0.4583333333333333, 'gini = 0.0\nsamples = 1\nvalue
= [1, 0]'),
```

```
 Text(0.07317073170731707, 0.4583333333333333, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.17073170731707318, 0.5416666666666666, 'x[1] <= 41.5\ngini =
0.142\nsamples = 26\nvalue = [24, 2]'),
 Text(0.12195121951219512, 0.4583333333333333, 'x[2] <= 74500.0\ngini =
0.083\nsamples = 23\nvalue = [22, 1]'),
 Text(0.0975609756097561, 0.375, 'gini = 0.0\nsamples = 12\nvalue = [12, 0]'),
 Text(0.14634146341463414, 0.375, 'x[2] <= 76000.0\ngini = 0.165\nsamples =
11\nvalue = [10, 1]'),
 Text(0.12195121951219512, 0.2916666666666667, 'x[0] <= 0.5\ngini =
0.375\nsamples = 4\nvalue = [3, 1]'),
 Text(0.0975609756097561, 0.20833333333333334, 'x[1] <= 39.5\ngini =
0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.07317073170731707, 0.125, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.12195121951219512, 0.125, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.14634146341463414, 0.20833333333333334, 'gini = 0.0\nsamples = 2\nvalue
= [2, 0]'),
 Text(0.17073170731707318, 0.2916666666666667, 'gini = 0.0\nsamples = 7\nvalue =
[7, 0]'),
 Text(0.21951219512195122, 0.4583333333333333, 'x[2] <= 74000.0\ngini =
0.444\nsamples = 3\nvalue = [2, 1]'),
 Text(0.1951219512195122, 0.375, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.24390243902439024, 0.375, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.3170731707317073, 0.7916666666666666, 'x[1] <= 26.5\ngini =
0.284\nsamples = 35\nvalue = [6, 29]'),
 Text(0.2926829268292683, 0.7083333333333334, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]'),
 Text(0.34146341463414637, 0.7083333333333334, 'x[2] <= 116500.0\ngini =
0.213\nsamples = 33\nvalue = [4, 29]'),
 Text(0.3170731707317073, 0.625, 'x[2] <= 107500.0\ngini = 0.408\nsamples =
14\nvalue = [4, 10]'),
 Text(0.2926829268292683, 0.5416666666666666, 'gini = 0.0\nsamples = 8\nvalue =
[0, 8]'),
 Text(0.34146341463414637, 0.5416666666666666, 'x[1] <= 36.5\ngini =
0.444\nsamples = 6\nvalue = [4, 2]'),
 Text(0.3170731707317073, 0.4583333333333333, 'x[2] <= 112500.0\ngini =
0.32\nsamples = 5\nvalue = [4, 1]'),
 Text(0.2926829268292683, 0.375, 'x[2] <= 110000.0\ngini = 0.5\nsamples =
2\nvalue = [1, 1]'),
 Text(0.2682926829268293, 0.2916666666666667, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(0.3170731707317073, 0.2916666666666667, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.34146341463414637, 0.375, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.36585365853658536, 0.4583333333333333, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.36585365853658536, 0.625, 'gini = 0.0\nsamples = 19\nvalue = [0, 19]'),
```

```
 Text(0.551829268292683, 0.875, 'x[2] <= 38500.0\ngini = 0.296\nsamples =
83\nvalue = [15, 68]'),
 Text(0.4634146341463415, 0.7916666666666666, 'x[2] <= 22500.0\ngini =
0.069\nsamples = 28\nvalue = [1, 27]'),
 Text(0.43902439024390244, 0.7083333333333334, 'x[1] <= 46.5\ngini =
0.375\nsamples = 4\nvalue = [1, 3]'),
 Text(0.4146341463414634, 0.625, 'x[1] <= 45.5\ngini = 0.5\nsamples = 2\nvalue =
[1, 1]'),
 Text(0.3902439024390244, 0.5416666666666666, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.43902439024390244, 0.5416666666666666, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(0.4634146341463415, 0.625, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.4878048780487805, 0.7083333333333334, 'gini = 0.0\nsamples = 24\nvalue =
[0, 24]'),
 Text(0.6402439024390244, 0.7916666666666666, 'x[2] <= 44500.0\ngini =
0.38\nsamples = 55\nvalue = [14, 41]'),
 Text(0.5365853658536586, 0.7083333333333334, 'x[2] <= 41500.0\ngini =
0.444\nsamples = 6\nvalue = [4, 2]'),
 Text(0.5121951219512195, 0.625, 'x[1] <= 45.0\ngini = 0.444\nsamples = 3\nvalue
= [1, 2]'),
 Text(0.4878048780487805, 0.5416666666666666, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(0.5365853658536586, 0.5416666666666666, 'gini = 0.0\nsamples = 2\nvalue =
[0, 2]'),
 Text(0.5609756097560976, 0.625, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.7439024390243902, 0.7083333333333334, 'x[1] <= 46.5\ngini =
0.325\nsamples = 49\nvalue = [10, 39]'),
 Text(0.6341463414634146, 0.625, 'x[2] <= 106500.0\ngini = 0.486\nsamples =
12\nvalue = [5, 7]'),
 Text(0.5853658536585366, 0.5416666666666666, 'x[2] <= 52000.0\ngini =
0.49\nsamples = 7\nvalue = [4, 3]'),
 Text(0.5609756097560976, 0.4583333333333333, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.6097560975609756, 0.4583333333333333, 'x[0] <= 0.5\ngini =
0.444\nsamples = 6\nvalue = [4, 2]'),
 Text(0.5853658536585366, 0.375, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.6341463414634146, 0.375, 'x[2] <= 69000.0\ngini = 0.5\nsamples =
4\nvalue = [2, 2]'),
 Text(0.6097560975609756, 0.2916666666666667, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(0.6585365853658537, 0.2916666666666667, 'x[1] <= 45.5\ngini =
0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.6341463414634146, 0.20833333333333334, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(0.6829268292682927, 0.20833333333333334, 'gini = 0.0\nsamples = 2\nvalue =
[0, 2]'),
```

```
 Text(0.6829268292682927, 0.5416666666666666, 'x[2] <= 132000.0\ngini =
0.32\nsamples = 5\nvalue = [1, 4]'),
 Text(0.6585365853658537, 0.4583333333333333, 'gini = 0.0\nsamples = 3\nvalue =
[0, 3]'),
 Text(0.7073170731707317, 0.4583333333333333, 'x[1] <= 43.5\ngini = 0.5\nsamples
= 2\nvalue = [1, 1]'),
 Text(0.6829268292682927, 0.375, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.7317073170731707, 0.375, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.8536585365853658, 0.625, 'x[1] <= 52.5\ngini = 0.234\nsamples =
37\nvalue = [5, 32]'),
 Text(0.7804878048780488, 0.5416666666666666, 'x[1] <= 47.5\ngini =
0.346\nsamples = 18\nvalue = [4, 14]'),
 Text(0.7560975609756098, 0.4583333333333333, 'gini = 0.0\nsamples = 6\nvalue =
[0, 6]'),
 Text(0.8048780487804879, 0.4583333333333333, 'x[2] <= 75500.0\ngini =
0.444\nsamples = 12\nvalue = [4, 8]'),
 Text(0.7804878048780488, 0.375, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.8292682926829268, 0.375, 'x[2] <= 102000.0\ngini = 0.397\nsamples =
11\nvalue = [3, 8]'),
 Text(0.8048780487804879, 0.2916666666666667, 'gini = 0.0\nsamples = 4\nvalue =
[0, 4]'),
 Text(0.8536585365853658, 0.2916666666666667, 'x[2] <= 136000.0\ngini =
0.49\nsamples = 7\nvalue = [3, 4]'),
 Text(0.8292682926829268, 0.20833333333333334, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]'),
 Text(0.8780487804878049, 0.20833333333333334, 'x[0] <= 0.5\ngini =
0.32\nsamples = 5\nvalue = [1, 4]'),
 Text(0.8536585365853658, 0.125, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.9024390243902439, 0.125, 'x[1] <= 50.0\ngini = 0.5\nsamples = 2\nvalue =
[1, 1]'),
 Text(0.8780487804878049, 0.041666666666666664, 'gini = 0.0\nsamples = 1\nvalue
= [1, 0]'),
 Text(0.926829268292683, 0.041666666666666664, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.926829268292683, 0.5416666666666666, 'x[1] <= 58.5\ngini = 0.1\nsamples
= 19\nvalue = [1, 18]'),
 Text(0.9024390243902439, 0.4583333333333333, 'gini = 0.0\nsamples = 13\nvalue =
[0, 13]'),
 Text(0.9512195121951219, 0.4583333333333333, 'x[2] <= 85500.0\ngini =
0.278\nsamples = 6\nvalue = [1, 5]'),
 Text(0.926829268292683, 0.375, 'x[1] <= 59.5\ngini = 0.5\nsamples = 2\nvalue =
[1, 1]'),
 Text(0.9024390243902439, 0.2916666666666667, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(0.9512195121951219, 0.2916666666666667, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.975609756097561, 0.375, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]')]
```



[13]:

```
#h. Prune the tree with maximum depth as 3,5,7 and tabulate the various TP, TN,␣
↪accuracy,
#f-score and AUC score obtained.
print(1128)

for i in [3,5,7]:
    tree_entropy = DecisionTreeClassifier(criterion='entropy', max_depth=i)
    tree_entropy.fit(x_train,y_train)
    y_pred = tree_entropy.predict(x_test)
    tree_ig = DecisionTreeClassifier(criterion='gini',max_depth=i)
    tree_ig.fit(x_train,y_train)
    y_pred_ig = tree_ig.predict(x_test)
    #Accuracy
    print('Accuracy using Entropy : ',accuracy_score(y_test,y_pred))
    print('Accuracy using ig : ',accuracy_score(y_test,y_pred_ig))
    #f1_score
    print('F1_score using Entropy : ',f1_score(y_test,y_pred))
    print('F1_score using ig : ',f1_score(y_test,y_pred_ig))
        #AUC score
    print('AUC score using Entropy : ',roc_auc_score(y_test,y_pred))
    print('AUC score using ig : ',roc_auc_score(y_test,y_pred_ig))
    conf_matrix=confusion_matrix(y_test,y_pred)
    conf_matrix1=confusion_matrix(y_test,y_pred_ig)
```

```
        #tp
    print('TP for entropy : ',conf_matrix[1][1])
    print('TP for ig : ',conf_matrix1[1][1])
        #tn
    print('TN for entropy : ',conf_matrix[0][0])
    print('TN for ig : ',conf_matrix1[0][0])
```

```
1128
Accuracy using Entropy :  0.89
Accuracy using ig :  0.85
F1_score using Entropy :  0.8735632183908046
F1_score using ig :  0.8192771084337349
AUC score using Entropy :  0.8892288861689106
AUC score using ig :  0.8427172582619339
TP for entropy :  38
TP for ig :  34
TN for entropy :  51
TN for ig :  51
Accuracy using Entropy :  0.83
Accuracy using ig :  0.88
F1_score using Entropy :  0.7733333333333334
F1_score using ig :  0.8536585365853658
AUC score using Entropy :  0.8108935128518971
AUC score using ig :  0.871889024887801
TP for entropy :  29
TP for ig :  35
TN for entropy :  54
TN for ig :  53
Accuracy using Entropy :  0.84
Accuracy using ig :  0.84
F1_score using Entropy :  0.8048780487804877
F1_score using ig :  0.8
AUC score using Entropy :  0.8310893512851898
AUC score using ig :  0.8282333741330069
TP for entropy :  33
TP for ig :  32
TN for entropy :  51
TN for ig :  52
```

[ ]:
```
RESULT:
    Thus the performance analysis for decision tree classification technique is␣
 ↪done successfully.
```

Ex.9 Clustering of Data using K-means Clustering Technique

October 26, 2023

[ ]:
```
Bewin Felix R A
URK21CS1128
```

[ ]:
```
AIM:
    To apply K-means clustering technique to the given dataset.
```

[ ]:
```
DESCRIPTION:

K-means clustering is a popular unsupervised machine learning
technique used to partition a dataset into K distinct, non-overlapping
clusters based on the similarity of data points. It aims to group similar
data points together and find cluster centroids that minimize the
within-cluster variance.
from sklearn.cluster import KMeans
# Choose the number of clusters (K)
k = 3
# Initialize the K-means model
kmeans = KMeans(n_clusters=k, init='k-means++', max_iter=300, n_init=10,
random_state=0)
# Fit the model to your data
kmeans.fit(X)
# Get cluster labels for each data point
labels = kmeans.labels_
# Get cluster centroids
centroids = kmeans.cluster_centers_
Elbow Method for Optimal K:
Description: The Elbow Method helps in determining the optimal number of
clusters (K) for K-means. It involves fitting K-means with different K
values and plotting the Within-Cluster Sum of Squares (WCSS) against K.
wcss = []
for k in range(1, 11):
kmeans = KMeans(n_clusters=k, init='k-means++', max_iter=300, n_init=10,
random_state=0)
kmeans.fit(X)
wcss.append(kmeans.inertia_)
# Plot WCSS vs. K
# ...
Visualization of Clusters:
```

```
Description: Visualizing clusters helps in understanding the clustering results.
 You can create scatter plots with data points colored by their cluster
assignments.
plt.scatter(X['Age'], X['Annual_Income'], c=kmeans.labels_, cmap='rainbow')
plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='black', marker='X',
label='Centroids')
plt.title('K-means Clustering')
plt.xlabel('Age')
plt.ylabel('Annual Income')
plt.legend()
plt.show()
6. Performance Metrics for Different K-values
Description: You calculate performance metrics (silhouette_score and
davies_bouldin_score) for different values of K to evaluate the quality of
clustering.
k_values = [optimal_k, optimal_k + 1, optimal_k + 2, optimal_k + 3]
silhouette_scores = []
davies_bouldin_scores = []
for k in k_values:
kmeans = KMeans(n_clusters=k, init='k-means++', max_iter=300, n_init=10,
                random_state=0)
kmeans.fit(X)
silhouette = silhouette_score(X, kmeans.labels_)
davies_bouldin = davies_bouldin_score(X, kmeans.labels_)
silhouette_scores.append(silhouette)
davies_bouldin_scores.append(davies_bouldin)
```

[ ]:

```
2. Develop a K-means clustering model for the Mall_Customers dataset using the␣
 ↪scikit-
learn.
```

[10]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, davies_bouldin_score
```

[11]:

```
print(1128)
df = pd.read_csv('Mall_Customers.csv')
df.head()
```

1128

[11]:

	CustomerID	Gender	Age	Annual_Income	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77

```
4           5  Female   31              17                       40
```

[12]:
```python
#a. Use the columns: ' Age', 'Annual_Income' as the input variables.
print(1128)
X = df[['Age', 'Annual_Income']]
X.head()
```

```
1128
```

[12]:

	Age	Annual_Income
0	19	15
1	21	15
2	20	16
3	23	16
4	31	17

[13]:
```python
# b. Compute the optimal number of cluster ‘K’ from 1-10 using the Elbow method.
print(1128)
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++',␣
 ↪max_iter=300,n_init=10,random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
```

```
1128
```

[ ]:
```python
#c. Plot the graph between number of cluster K and within-cluster sum of␣
 ↪squares (WCSS)
#value.
print(1128)
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS')
plt.grid()
plt.show()
```

[14]:
```python
# d. Perform the K-means clustering with the selected optimal K.
print(1128)
optimal_k = 4

kmeans = KMeans(n_clusters=optimal_k, init='k-means++',␣
 ↪max_iter=300,n_init=10,random_state=0)
kmeans.fit(X)
```

```
1128
```

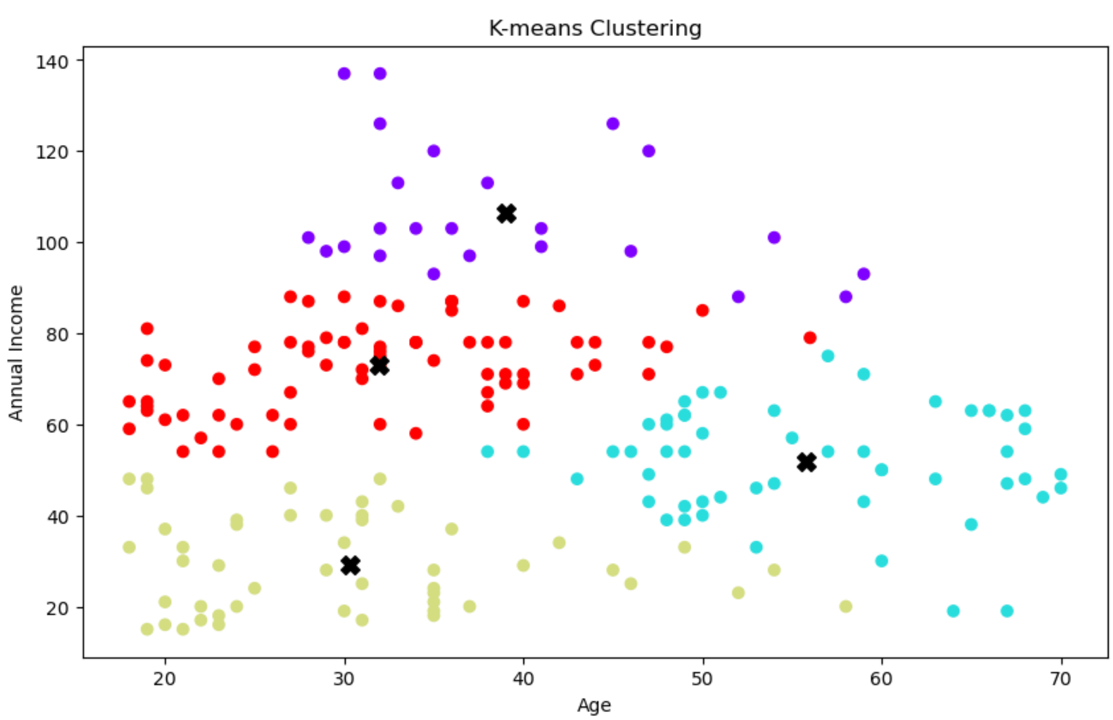
[14]: KMeans(n_clusters=4, n_init=10, random_state=0)

[15]:
```
# e. Display the cluster centroids.
print(1128)
centroids = kmeans.cluster_centers_
print("Cluster Centroids:")
print(centroids)
```

```
1128
Cluster Centroids:
[[ 39.         106.5       ]
 [ 55.81481481  51.77777778]
 [ 30.34693878  29.26530612]
 [ 31.95890411  72.95890411]]
```

[16]:
```
# f. Visualize the data representation of K-means clustering.
print(1128)
df['Cluster'] = kmeans.labels_
plt.figure(figsize=(10, 6))
plt.scatter(df['Age'], df['Annual_Income'], c=df['Cluster'], cmap='rainbow')
plt.scatter(centroids[:, 0], centroids[:, 1], c='black', marker='X', s=100)
plt.title('K-means Clustering')
plt.xlabel('Age')
plt.ylabel('Annual Income')
plt.show()
```

```
1128
```



[17]:
```python
# g. Change the value of K in K-means with different values and tabulate the␣
↪performance
#metrics such as silhouette_score and davies_bouldin_score obtained.
print(1128)
k_values_to_evaluate = [optimal_k, optimal_k + 1, optimal_k + 2, optimal_k + 3]
print("K-value\tSilhouette Score\tDavies-Bouldin Score")
for k in k_values_to_evaluate:
    kmeans = KMeans(n_clusters=k, init='k-means++', max_iter=300,
                    n_init=10,random_state=0)
    kmeans.fit(X)
    silhouette = silhouette_score(X, kmeans.labels_)
    davies_bouldin = davies_bouldin_score(X, kmeans.labels_)
    print(f"{k}\t{silhouette:.4f}\t\t{davies_bouldin:.4f}")
```

```
1128
K-value Silhouette Score        Davies-Bouldin Score
4       0.4330          0.7696
5       0.4084          0.7552
6       0.3955          0.8186
7       0.3847          0.8472
```

[ ]:
```
RESULT:
    Therefore the K-means clustering technique has been applied to the
    dataset given and the output is displayed successfully.
```

Ex.10 Design of Content-based Recommender System

October 26, 2023

[ ]:
```
Bewin Felix R A
URK21CS1128
```

[ ]:
```
AIM:
    To Develop an E-commerce item recommender system with content-based␣
 ↪recommendation system
using the Term-Frequency Inverse Document Frequency (TF IDF) and cosine␣
 ↪similarity.
```

[ ]:
```
DESCRIPTION:

A recommendation system, often referred to as a recommender system, is a type of
software or algorithm that provides personalized suggestions to users. These
suggestions can be for products, services, content, or items of interest based␣
 ↪on
a user's past behavior, preferences, or the behavior of similar users.
Recommendation systems are widely used in various domains, including e-commerce,
content streaming, social media, and more, to enhance user experience and␣
 ↪engagement.

A search engine is a software application or online service that helps users␣
 ↪find
information on the internet or within a specific dataset. It works by indexing␣
 ↪and
cataloging vast amounts of web content and providing users with a way to search␣
 ↪for
and access relevant information quickly.

Term Frequency-Inverse Document Frequency, commonly abbreviated as TF-IDF, is a
numerical statistic used in information retrieval and text mining to evaluate
the importance of a word in a document relative to a collection of documents
(corpus). TF-IDF is a technique that combines two key components:

Term Frequency (TF):

Term Frequency measures how frequently a term (word or phrase) appears in a
document. It is calculated as the number of times a term occurs in a document
```

```
divided by the total number of terms in the document. The idea is to give higher
weight to terms that appear more frequently within a document.

Inverse Document Frequency (IDF):

Inverse Document Frequency calculates the importance of a term across a␣
↪collection
of documents. It is used to downweight common terms that appear in many␣
↪documents
and give higher weight to rare terms.

The TF-IDF score for a term in a document combines both the TF and IDF␣
↪components
to determine how important the term is in that particular document within the
context of the entire corpus. It is calculated as follows:

TF-IDF(t, d, corpus) = TF(t, d) * IDF(t, corpus)

# from sklearn.feature_extraction.text import TfidfVectorizer
# tfidf = TfidfVectorizer(stop_words='english')
# tfidf_matrix = tfidf.fit_transform(content)

Cosine similarity
Cosine similarity measures the similarity between two vectors. Since TF-IDF␣
↪returns
vectors showing the score a document gets versus the corpus, we can use cosine
similarity to identify the closest matches after we’ve used TF-IDF to generate␣
↪the
vectors.
# from sklearn.metrics.pairwise import cosine_similarity
# from sklearn.metrics.pairwise import linear_kernel
```

[ ]:
```
2. Develop an E-commerce item recommender system with content-based␣
↪recommendation
using the scikit-learn
a. Use the column: 'product'.
```

[1]:
```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics.pairwise import linear_kernel
```

[3]:
```
print(1128)
df = pd.read_csv("shop_details.csv")
df
```

```
1128
```

[3]:
```
        id                    product  \
0      1.0      Active classic boxers
1      2.0  Active sport boxer briefs
2      3.0        Active sport briefs
3      4.0         Alpine guide pants
4      5.0            Alpine wind jkt
…    …                          …
2182   NaN                        NaN
2183   NaN                        NaN
2184   NaN                        NaN
2185   NaN                        NaN
2186   NaN                        NaN

                                             description
0      There's a reason why our boxers are a cult fav…
1      Skinning up Glory requires enough movement wit…
2      These superbreathable no-fly briefs are the mi…
3      Skin in, climb ice, switch to rock, traverse a…
4      On high ridges, steep ice and anything alpine,…
…                                                   …
2182                                               NaN
2183                                               NaN
2184                                               NaN
2185                                               NaN
2186                                               NaN

[2187 rows x 3 columns]
```

[4]:
```
#b. Remove the leading and trailing whitespaces in that column.
print(1128)
df["product"].str.strip()
df["product"]
```

```
1128
```

[4]:
```
0           Active classic boxers
1       Active sport boxer briefs
2             Active sport briefs
3              Alpine guide pants
4                 Alpine wind jkt
                    …
2182                          NaN
2183                          NaN
2184                          NaN
2185                          NaN
2186                          NaN
```

```
Name: product, Length: 2187, dtype: object
```

[5]:
```
#removing NaN values from the dataset.
print(1128)
df.dropna(inplace=True)
df
```

```
1128
```

[5]:
```
        id                 product  \
0      1.0      Active classic boxers
1      2.0  Active sport boxer briefs
2      3.0        Active sport briefs
3      4.0         Alpine guide pants
4      5.0           Alpine wind jkt
..     …                         …
495  496.0             Cap 2 bottoms
496  497.0                Cap 2 crew
497  498.0             All-time shell
498  499.0      All-wear cargo shorts
499  500.0           All-wear shorts

                                           description
0    There's a reason why our boxers are a cult fav…
1    Skinning up Glory requires enough movement wit…
2    These superbreathable no-fly briefs are the mi…
3    Skin in, climb ice, switch to rock, traverse a…
4    On high ridges, steep ice and anything alpine,…
..                                                …
495  Cut loose from the maddening crowds and search…
496  This crew takes the edge off fickle weather. I…
497  No need to use that morning Times as an umbrel…
498  All-Wear Cargo Shorts bask in the glory of swe…
499  Time to simplify? Our All-Wear shorts prove th…

[500 rows x 3 columns]
```

[6]:
```
df.columns
```

[6]:
```
Index(['id', 'product', 'description'], dtype='object')
```

[7]:
```
# c. Perform feature extraction using Term Frequency Inverse Document Frequency␣
↪(TF-
#IDF).
print(1128)
indices = pd.Series(df.index, index=df['product']).drop_duplicates()
content = df['description'].fillna('')
```

```python
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(content)
print("TF IDF Matrix:",tfidf_matrix)
```

```
1128
TF IDF Matrix:   (0, 2550)	0.10398386495339977
  (0, 294)	0.16071014411893011
  (0, 4478)	0.026691279623776477
  (0, 1804)	0.08733693134440922
  (0, 1057)	0.09201923404633659
  (0, 2741)	0.08577597087914687
  (0, 2665)	0.08843160800635147
  (0, 1836)	0.09471579837759175
  (0, 979)	0.06091036994464541
  (0, 1433)	0.0734929307585737
  (0, 1379)	0.0644741003780651
  (0, 703)	0.06492701570977669
  (0, 4256)	0.11673739714671233
  (0, 1569)	0.05253687697191237
  (0, 793)	0.08957389399439541
  (0, 3570)	0.10959412306927277
  (0, 2356)	0.2101475078876495
  (0, 4251)	0.05253687697191237
  (0, 1266)	0.026268438485956187
  (0, 695)	0.2101475078876495
  (0, 3052)	0.07093203632068933
  (0, 3178)	0.07093203632068933
  (0, 4077)	0.07093203632068933
  (0, 989)	0.07093203632068933
  (0, 3176)	0.07093203632068933
  :	:
  (499, 4315)	0.10777030548461138
  (499, 4099)	0.09192482716078833
  (499, 2979)	0.2243765958317256
  (499, 1391)	0.1007954104508571
  (499, 1672)	0.07744459615973451
  (499, 3690)	0.0486578839623734
  (499, 4478)	0.027391812019129633
  (499, 1804)	0.08962915376985607
  (499, 1569)	0.026957873102585114
  (499, 2356)	0.3774102234361916
  (499, 4251)	0.05391574620517023
  (499, 1266)	0.026957873102585114
  (499, 695)	0.21566298482068091
  (499, 3052)	0.07279370013042086
  (499, 3178)	0.07279370013042086
  (499, 4077)	0.07279370013042086
  (499, 989)	0.07279370013042086
```

```
  (499, 3176)	0.07279370013042086
  (499, 3595)	0.07094338003490643
  (499, 2155)	0.1392038311496402
  (499, 3)	0.0817450340489487
  (499, 2811)	0.05478362403825927
  (499, 1714)	0.07235510954312023
  (499, 1655)	0.05722767997477116
  (499, 2370)	0.05625319291837664
```

[8]:
```python
# d. Compute the cosine similarity.
print(1128)
cosine_similarities = linear_kernel(tfidf_matrix, tfidf_matrix)
print(cosine_similarities)
```

```
1128
[[1.         0.279554   0.19517104 … 0.15671646 0.18352571 0.21493076]
 [0.279554   1.         0.54659561 … 0.12053582 0.22053005 0.19737709]
 [0.19517104 0.54659561 1.         … 0.1051828  0.12856782 0.15275201]
 …
 [0.15671646 0.12053582 0.1051828  … 1.         0.11754784 0.14264239]
 [0.18352571 0.22053005 0.12856782 … 0.11754784 1.         0.57147933]
 [0.21493076 0.19737709 0.15275201 … 0.14264239 0.57147933 1.        ]]
```

[9]:
```python
# e. Display the top 'n' suggestions with the similarity score for the given␣
 ↪user input.
print(1128)
def get_recommendations(df, column, value, cosine_similarities, n):



    # Return indices for the target dataframe column and drop any duplicates
    indices = pd.Series(df.index, index=df[column]).drop_duplicates()

    # Get the index for the target value
    target_index = indices[value]

    # Get the cosine similarity scores for the target value
    cosine_similarity_scores =␣
 ↪list(enumerate(cosine_similarities[target_index]))

    # Sort the cosine similarities in order of closest similarity
    cosine_similarity_scores = sorted(cosine_similarity_scores, key=lambda x:␣
 ↪x[1], reverse=True)

    # Return tuple of the requested closest scores excluding the target item␣
 ↪and index
    cosine_similarity_scores = cosine_similarity_scores[1:n+1]

    # Extract the tuple values
```

```
    index = (x[0] for x in cosine_similarity_scores)
    scores = (x[1] for x in cosine_similarity_scores)

    # Get the indices for the closest items
    recommendation_indices = [i[0] for i in cosine_similarity_scores]

    # Get the actutal recommendations
    recommendations = df[column].iloc[recommendation_indices]

    # Return a dataframe
    df = pd.DataFrame(list(zip(index, recommendations, scores)),
                      columns=['index','recommendation',␣
 ↪'cosine_similarity_score'])

    return df

n = int(input("Enter the no.of suggetions:"))
recommendations = get_recommendations(df,
                                      'product',
                                      'Flying fish t-shirt',
                                      cosine_similarities,n)
```

```
1128
```

```
Enter the no.of suggetions: 2
```

[10]:
```
print(1128)
recommendations.head(10)
```

```
1128
```

[10]:

	index	recommendation	cosine_similarity_score
0	57	'73 logo t-shirt	0.911366
1	63	Gpiw classic t-shirt	0.892282

[ ]:
```
RESULT:
    The E-commerce item recommender system with content-based recommendation
using scikit-learn methods has been developed and the output is displayed␣
 ↪successfully.
```